ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/314354796

Data Noising as Smoothing in Neural Network Language Models

Article · March 2017

CITATIONS
9

READS
70

7 authors, including:

Daniel Levy
Stanford University
5 PUBLICATIONS **51** CITATIONS
SEE PROFILE

Dan Jurafsky
Stanford University
146 PUBLICATIONS **5,537** CITATIONS
SEE PROFILE

All content following this page was uploaded by Daniel Levy on 04 November 2018.

The user has requested enhancement of the downloaded file.

DATA NOISING AS SMOOTHING IN NEURAL NETWORK LANGUAGE MODELS

Ziang Xie, Sida I. Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, Andrew Y. Ng
Computer Science Department, Stanford University
{zxie,sidaw,danilevy,anie,ang}@cs.stanford.edu,
{jiweil,jurafsky}@stanford.edu

arXiv:1703.02573v1 [cs.LG] 7 Mar 2017

ABSTRACT

Data noising is an effective technique for regularizing neural network models. While noising is widely adopted in application domains such as vision and speech, commonly used noising primitives have not been developed for discrete sequence-level settings such as language modeling. In this paper, we derive a connection between input noising in neural network language models and smoothing in n-gram models. Using this connection, we draw upon ideas from smoothing to develop effective noising schemes. We demonstrate performance gains when applying the proposed schemes to language modeling and machine translation. Finally, we provide empirical analysis validating the relationship between noising and smoothing.

1 INTRODUCTION

Language models are a crucial component in many domains, such as autocompletion, machine translation, and speech recognition. A key challenge when performing estimation in language modeling is the *data sparsity* problem: due to large vocabulary sizes and the exponential number of possible contexts, the majority of possible sequences are rarely or never observed, even for very short subsequences.

In other application domains, data augmentation has been key to improving the performance of neural network models in the face of insufficient data. In computer vision, for example, there exist well-established primitives for synthesizing additional image data, such as by rescaling or applying affine distortions to images (LeCun et al., 1998; Krizhevsky et al., 2012). Similarly, in speech recognition adding a background audio track or applying small shifts along the time dimension has been shown to yield significant gains, especially in noisy settings (Deng et al., 2000; Hannun et al., 2014). However, widely-adopted noising primitives have not yet been developed for neural network language models.

Classic n-gram models of language cope with rare and unseen sequences by using smoothing methods, such as interpolation or absolute discounting (Chen & Goodman, 1996). Neural network models, however, have no notion of discrete counts, and instead use distributed representations to combat the curse of dimensionality (Bengio et al., 2003). Despite the effectiveness of distributed representations, overfitting due to data sparsity remains an issue. Existing regularization methods, however, are typically applied to weights or hidden units within the network (Srivastava et al., 2014; Le et al., 2015) instead of directly considering the input data.

In this work, we consider noising primitives as a form of data augmentation for recurrent neural network-based language models. By examining the expected pseudocounts from applying the noising schemes, we draw connections between noising and linear interpolation smoothing. Using this connection, we then derive noising schemes that are analogues of more advanced smoothing methods. We demonstrate the effectiveness of these schemes for regularization through experiments on language modeling and machine translation. Finally, we validate our theoretical claims by examining the empirical effects of noising.

2 RELATED WORK

Our work can be viewed as a form of data augmentation, for which to the best of our knowledge there exists no widely adopted schemes in language modeling with neural networks. Classical regularization methods such as L_2-regularization are typically applied to the model parameters, while dropout is applied to activations which can be along the forward as well as the recurrent directions (Zaremba et al., 2014; Semeniuta et al., 2016; Gal, 2015). Others have introduced methods for recurrent neural networks encouraging the hidden activations to remain stable in norm, or constraining the recurrent weight matrix to have eigenvalues close to one (Krueger & Memisevic, 2015; Arjovsky et al., 2015; Le et al., 2015). These methods, however, all consider weights and hidden units instead of the input data, and are motivated by the vanishing and exploding gradient problem.

Feature noising has been demonstrated to be effective for structured prediction tasks, and has been interpreted as an explicit regularizer (Wang et al., 2013). Additionally, Wager et al. (2014) show that noising can inject appropriate generative assumptions into discriminative models to reduce their generalization error, but do not consider sequence models (Wager et al., 2016).

The technique of randomly zero-masking input word embeddings for learning sentence representations has been proposed by Iyyer et al. (2015), Kumar et al. (2015), and Dai & Le (2015), and adopted by others such as Bowman et al. (2015). However, to the best of our knowledge, no analysis has been provided besides reasoning that zeroing embeddings may result in a model ensembling effect similar to that in standard dropout. This analysis is applicable to classification tasks involving sum-of-embeddings or bag-of-words models, but does not capture sequence-level effects. Bengio et al. (2015) also make an empirical observation that the method of randomly replacing words with fixed probability with a draw from the uniform distribution improved performance slightly for an image captioning task; however, they do not examine why performance improved.

3 METHOD

3.1 PRELIMINARIES

We consider language models where given a sequence of indices $X = (x_1, x_2, \cdots, x_T)$, over the vocabulary V, we model

$$p(X) = \prod_{t=1}^{T} p(x_t | x_{<t})$$

In n-gram models, it is not feasible to model the full context $x_{<t}$ for large t due to the exponential number of possible histories. Recurrent neural network (RNN) language models can (in theory) model longer dependencies, since they operate over distributed hidden states instead of modeling an exponential number of discrete counts (Bengio et al., 2003; Mikolov, 2012).

An L-layer recurrent neural network is modeled as $h_t^{(l)} = f_\theta(h_{t-1}^{(l)}, h_t^{(l-1)})$, where l denotes the layer index, $h^{(0)}$ contains the one-hot encoding of X, and in its simplest form f_θ applies an affine transformation followed by a nonlinearity. In this work, we use RNNs with a more complex form of f_θ, namely long short-term memory (LSTM) units (Hochreiter & Schmidhuber, 1997), which have been shown to ease training and allow RNNs to capture longer dependencies. The output distribution over the vocabulary V at time t is $p_\theta(x_t|x_{<t}) = \mathrm{softmax}(g_\theta(h_t^{(L)}))$, where $g : \mathbb{R}^{|h|} \to \mathbb{R}^{|V|}$ applies an affine transformation. The RNN is then trained by minimizing over its parameters θ the sequence cross-entropy loss $\ell(\theta) = -\sum_t \log p_\theta(x_t|x_{<t})$, thus maximizing the likelihood $p_\theta(X)$.

As an extension, we also consider encoder-decoder or sequence-to-sequence (Cho et al., 2014; Sutskever et al., 2014) models where given an input sequence X and output sequence Y of length T_Y, we model

$$p(Y|X) = \prod_{t=1}^{T_Y} p(y_t | X, y_{<t}).$$

and minimize the loss $\ell(\theta) = -\sum_t \log p_\theta(y_t|X, y_{<t})$. This setting can also be seen as conditional language modeling, and encompasses tasks such as machine translation, where X is a source lan-

guage sequence and Y a target language sequence, as well as language modeling, where Y is the given sequence and X is the empty sequence.

3.2 SMOOTHING AND NOISING

Recall that for a given context length l, an n-gram model of order $l+1$ is optimal under the log-likelihood criterion. Hence in the case where an RNN with finite context achieves near the lowest possible cross-entropy loss, it behaves like an n-gram model.

Like n-gram models, RNNs are trained using maximum likelihood, and can easily overfit (Zaremba et al., 2014). While generic regularization methods such L_2-regularization and dropout are effective, they do not take advantage of specific properties of sequence modeling. In order to understand sequence-specific regularization, it is helpful to examine n-gram language models, whose properties are well-understood.

Smoothing for n-gram models When modeling $p(x_t|x_{<t})$, the maximum likelihood estimate $c(x_{<t}, x_t)/c(x_{<t})$ based on empirical counts puts zero probability on unseen sequences, and thus smoothing is crucial for obtaining good estimates. In particular, we consider interpolation, which performs a weighted average between higher and lower order models. The idea is that when there are not enough observations of the full sequence, observations of subsequences can help us obtain better estimates.[1] For example, in a bigram model, $p_{\text{interp}}(x_t|x_{t-1}) = \lambda p(x_t|x_{t-1}) + (1-\lambda)p(x_t)$, where $0 \leq \lambda \leq 1$.

Noising for RNN models We would like to apply well-understood smoothing methods such as interpolation to RNNs, which are also trained using maximum likelihood. Unfortunately, RNN models have no notion of counts, and we cannot directly apply one of the usual smoothing methods. In this section, we consider two simple noising schemes which we proceed to show correspond to smoothing methods. Since we can noise the data while training an RNN, we can then incorporate well-understood generative assumptions that are known to be helpful in the domain. First consider the following two noising schemes:

- **unigram noising** For each x_i in $x_{<t}$, with probability γ replace x_i with a sample from the unigram frequency distribution.
- **blank noising** For each x_i in $x_{<t}$, with probability γ replace x_i with a placeholder token "_".

While blank noising can be seen as a way to avoid overfitting on specific contexts, we will see that both schemes are related to smoothing, and that unigram noising provides a path to analogues of more advanced smoothing methods.

3.3 NOISING AS SMOOTHING

We now consider the maximum likelihood estimate of n-gram probabilities estimated using the pseudocounts of the noised data. By examining these estimates, we draw a connection between linear interpolation smoothing and noising.

Unigram noising as interpolation To start, we consider the simplest case of bigram probabilities. Let $c(x)$ denote the count of a token x in the original data, and let $c_\gamma(x) \stackrel{\text{def}}{=} \mathbb{E}_{\tilde{x}}\left[c(\tilde{x})\right]$ be the expected count of x under the unigram noising scheme. We then have

$$\begin{aligned} p_\gamma(x_t|x_{t-1}) &= \frac{c_\gamma(x_{t-1}, x_t)}{c_\gamma(x_{t-1})} \\ &= [(1-\gamma)c(x_{t-1}, x_t) + \gamma\, p(x_{t-1})c(x_t)]/c(x_{t-1}) \\ &= (1-\gamma)p(x_t|x_{t-1}) + \gamma\, p(x_t), \end{aligned}$$

where $c_\gamma(x) = c(x)$ since our proposal distribution $q(x)$ is the unigram distribution, and the last line follows since $c(x_{t-1})/p(x_{t-1}) = c(x_t)/p(x_t)$ is equal to the total number of tokens in the training

[1] For a thorough review of smoothing methods, we defer to Chen & Goodman (1996).

set. Thus we see that the noised data has pseudocounts corresponding to *interpolation* or a mixture of different order n-gram models with fixed weighting.

More generally, let $\tilde{x}_{<t}$ be noised tokens from $\tilde{x}$. We consider the expected prediction under noise

$$\begin{aligned} p_\gamma(x_t|x_{<t}) &= \mathbb{E}_{\tilde{x}_{<t}}\left[p(x_t|\tilde{x}_{<t})\right] \\ &= \sum_J \underbrace{\pi(|J|)}_{p(|J| \text{ swaps})} \sum_{x_K} \underbrace{p(x_t|x_J, x_K)}_{p(x_t|\text{noised context})} \prod_{z \in x_K} \underbrace{p(z)}_{p(\text{drawing } z)} \end{aligned}$$

where the mixture coefficients are $\pi(|J|) = (1-\gamma)^{|J|}\gamma^{t-1-|J|}$ with $\sum_J \pi(|J|) = 1$. $J \subseteq \{1, 2, \ldots, t-1\}$ denotes the set of indices whose corresponding tokens are left unchanged, and K the set of indices that were replaced.

Blank noising as interpolation Next we consider the blank noising scheme and show that it corresponds to interpolation as well. This also serves as an alternative explanation for the gains that other related work have found with the "word-dropout" idea (Kumar et al., 2015; Dai & Le, 2015; Bowman et al., 2015). As before, we do not noise the token being predicted x_t. Let $\tilde{x}_{<t}$ denote the random variable where each of its tokens is replaced by "_" with probability γ, and let x_J denote the sequence with indices J unchanged, and the rest replaced by "_". To make a prediction, we use the expected probability over different noisings of the context

$$p_\gamma(x_t|x_{<t}) = \mathbb{E}_{\tilde{x}_{<t}}\left[p(x_t|\tilde{x}_{<t})\right] = \sum_J \underbrace{\pi(|J|)}_{p(|J| \text{ swaps})} \underbrace{p(x_t|x_J)}_{p(x_t|\text{noised context})} \quad ,$$

where $J \subseteq \{1, 2, \ldots, t-1\}$, which is also a mixture of the unnoised probabilities over subsequences of the current context. For example, in the case of trigrams, we have

$$p_\gamma(x_3|x_1, x_2) = \pi(2)\, p(x_3|x_1, x_2) + \pi(1)\, p(x_3|x_1, \_) + \pi(1)\, p(x_3|\_, x_2) + \pi(0)\, p(x_3|\_, \_)$$

where the mixture coefficient $\pi(i) = (1-\gamma)^i\gamma^{2-i}$.

3.4 Borrowing Techniques

With the connection between noising and smoothing in place, we now consider how we can improve the two components of the noising scheme by considering:

1. Adaptively computing noising probability γ to reflect our confidence about a particular input subsequence.
2. Selecting a proposal distribution $q(x)$ that is less naive than the unigram distribution by leveraging higher order n-gram statistics.

Noising Probability Although it simplifies analysis, there is no reason why we should choose fixed γ; we now consider defining an adaptive $\gamma(x_{1:t})$ which depends on the input sequence. Consider the following bigrams:

"and the" "Humpty Dumpty"

The first bigram is one of the most common in English corpora; its probability is hence well estimated and should not be interpolated with lower order distributions. In expectation, however, using fixed γ_0 when noising results in the same lower order interpolation weight π_{γ_0} for common as well as rare bigrams. Intuitively, we should define $\gamma(x_{1:t})$ such that commonly seen bigrams are less likely to be noised.

The second bigram, "Humpty Dumpty," is relatively uncommon, as are its constituent unigrams. However, it forms what Brown et al. (1992) term a "sticky pair": the unigram "Dumpty" almost always follows the unigram "Humpty", and similarly, "Humpty" almost always precedes "Dumpty". For pairs with high mutual information, we wish to avoid backing off from the bigram to the unigram distribution.

Noised	$\gamma(x_{1:2})$	$q(x)$	Analogue
x_1	γ_0	$q(\text{“\_”}) = 1$	interpolation
x_1	γ_0	unigram	interpolation
x_1	$\gamma_0 N_{1+}(x_1, \bullet)/c(x_1)$	unigram	absolute discounting
x_1, x_2	$\gamma_0 N_{1+}(x_1, \bullet)/c(x_1)$	$q(x) \propto N_{1+}(\bullet, x)$	Kneser-Ney

Table 1: **Noising schemes** Example noising schemes and their bigram smoothing analogues. Here we consider the bigram probability $p(x_1, x_2) = p(x_2|x_1)p(x_1)$. Notation: $\gamma(x_{1:t})$ denotes the noising probability for a given input sequence $x_{1:t}$, $q(x)$ denotes the proposal distribution, and $N_{1+}(x, \bullet)$ denotes the number of distinct bigrams in the training set where x is the first unigram. In all but the last case we only noise the context x_1 and not the target prediction x_2.

Let $N_{1+}(x_1, \bullet) \stackrel{\text{def}}{=} |\{x_2 : c(x_1, x_2) > 0\}|$ be the number of distinct continutions following x_1, or equivalently the number of bigram types beginning with x_1 (Chen & Goodman, 1996). From the above intuitions, we arrive at the *absolute discounting* noising probability

$$\gamma_{\text{AD}}(x_1) = \gamma_0 \frac{N_{1+}(x_1, \bullet)}{\sum_{x_2} c(x_1, x_2)}$$

where for $0 \leq \gamma_0 \leq 1$ we have $0 \leq \gamma_{\text{AD}} \leq 1$, though in practice we can also clip larger noising probabilities to 1. Note that this encourages noising of unigrams that precede many possible other tokens while discouraging noising of common unigrams, since if we ignore the final token, $\sum_{x_2} c(x_1, x_2) = c(x_1)$.

Proposal Distribution While choosing the unigram distribution as the proposal distribution $q(x)$ preserves unigram frequencies, by borrowing from the smoothing literature we find another distribution performs better. We again begin with two motivating examples:

“San Francisco” “New York”

Both bigrams appear frequently in text corpora. As a direct consequence, the unigrams “Francisco” and “York” also appear frequently. However, since “Francisco” and “York” typically follow “San” and “New”, respectively, they should not have high probability in the proposal distribution as they might if we use unigram frequencies (Chen & Goodman, 1996). Instead, it would be better to increase the proposal probability of unigrams with diverse histories, or more precisely unigrams that complete a large number of bigram types. Thus instead of drawing from the unigram distribution, we consider drawing from

$$q(x) \propto N_{1+}(\bullet, x)$$

Note that we now noise the prediction x_t in addition to the context $x_{1:t-1}$. Combining this new proposal distribution with the discounted $\gamma_{\text{AD}}(x_1)$ from the previous section, we obtain the noising analogue of Kneser-Ney smoothing.

Table 1 summarizes the discussed noising schemes.

3.5 Training and Testing

During training, noising is performed per batch and is done online such that each epoch of training sees a different noised version of the training data. At test time, to match the training objective we should sample multiple corrupted versions of the test data, then average the predictions (Srivastava et al., 2014). In practice, however, we find that simply using the maximum likelihood (uncorrupted) input sequence works well; evaluation runtime remains unchanged.

3.6 Extensions

The schemes described are for the language model setting. To extend them to the sequence-to-sequence or encoder-decoder setting, we noise both $x_{<t}$ as well as $y_{<t}$. While in the decoder we

Noising scheme	Validation	Test
Medium models (512 hidden size)		
none (dropout only)	84.3	80.4
blank	82.7	78.8
unigram	83.1	80.1
bigram Kneser-Ney	**79.9**	**76.9**
Large models (1500 hidden size)		
none (dropout only)	81.6	77.5
blank	79.4	75.5
unigram	79.4	76.1
bigram Kneser-Ney	**76.2**	**73.4**
Zaremba et al. (2014)	82.2	78.4
Gal (2015) variational dropout (tied weights)	77.3	75.0
Gal (2015) (untied weights, Monte Carlo)	—	**73.4**

Table 2: Single-model perplexity on Penn Treebank with different noising schemes. We also compare to the variational method of Gal (2015), who also train LSTM models with the same hidden dimension. Note that performing Monte Carlo dropout at test time is significantly more expensive than our approach, where test time is unchanged.

Noising scheme	Validation	Test
none	94.3	123.6
blank	85.0	110.7
unigram	85.2	111.3
bigram Kneser-Ney	84.5	110.6

Table 3: Perplexity on Text8 with different noising schemes.

have $y_{<t}$ and y_t as analogues to language model context and target prediction, it is unclear whether noising $x_{<t}$ should be beneficial. Empirically, however, we find this to be the case (Table 4).

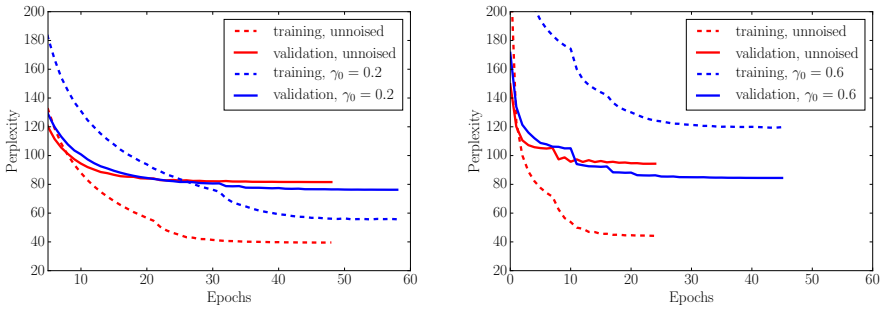
4 Experiments

4.1 Language Modeling

Penn Treebank We train networks for word-level language modeling on the Penn Treebank dataset, using the standard preprocessed splits with a 10K size vocabulary (Mikolov, 2012). The PTB dataset contains 929k training tokens, 73k validation tokens, and 82k test tokens. Following Zaremba et al. (2014), we use minibatches of size 20 and unroll for 35 time steps when performing backpropagation through time. All models have two hidden layers and use LSTM units. Weights are initialized uniformly in the range $[-0.1, 0.1]$. We consider models with hidden sizes of 512 and 1500.

We train using stochastic gradient descent with an initial learning rate of 1.0, clipping the gradient if its norm exceeds 5.0. When the validation cross entropy does not decrease after a training epoch, we halve the learning rate. We anneal the learning rate 8 times before stopping training, and pick the model with the lowest perplexity on the validation set.

For regularization, we apply feed-forward dropout (Pham et al., 2014) in combination with our noising schemes. We report results in Table 2 for the best setting of the dropout rate (which we find to match the settings reported in Zaremba et al. (2014)) as well as the best setting of noising



(a) Penn Treebank corpus.

(b) Text8 corpus.

Figure 1: Example training and validation curves for an unnoised model and model regularized using the bigram Kneser-Ney noising scheme.

Scheme	Perplexity	BLEU
dropout, no noising	8.84	24.6
blank noising	8.28	25.3 (+0.7)
unigram noising	8.15	25.5 (+0.9)
bigram Kneser-Ney	**7.92**	**26.0** (+1.4)
source only	8.74	24.8 (+0.2)
target only	8.14	25.6 (+1.0)

Table 4: Perplexities and BLEU scores for machine translation task. Results for bigram KN noising on only the source sequence and only the target sequence are given as well.

probability γ_0 on the validation set.[2] Figure 1 shows the training and validation perplexity curves for a noised versus an unnoised run.

Our large models match the state-of-the-art regularization method for single model performance on this task. In particular, we find that picking $\gamma_{\text{AD}}(x_1)$ and $q(x)$ corresponding to Kneser-Ney smoothing yields significant gains in validation perplexity, both for the medium and large size models. Recent work (Merity et al., 2016; Zilly et al., 2016) has also achieved impressive results on this task by proposing different architectures which are orthogonal to our data augmentation schemes.

Text8 In order to determine whether noising remains effective with a larger dataset, we perform experiments on the Text8 corpus[3]. The first 90M characters are used for training, the next 5M for validation, and the final 5M for testing, resulting in 15.3M training tokens, 848K validation tokens, and 855K test tokens. We preprocess the data by mapping all words which appear 10 or fewer times to the unknown token, resulting in a 42K size vocabulary. Other parameter settings are the same as described in the Penn Treebank experiments, besides that only models with hidden size 512 are considered, and noising is not combined with feed-forward dropout. Results are given in Table 3.

4.2 Machine Translation

For our machine translation experiments we consider the English-German machine translation track of IWSLT 2015[4]. The IWSLT 2015 corpus consists of sentence-aligned subtitles of TED and TEDx talks. The training set contains roughly 190K sentence pairs with 5.4M tokens. Following Luong & Manning (2015), we use TED tst2012 as a validation set and report BLEU score results (Papineni et al., 2002) on tst2014. We limit the vocabulary to the top 50K most frequent words for each language.

[2]Code will be made available at: `http://deeplearning.stanford.edu/noising`

[3]`http://mattmahoney.net/dc/text8.zip`

[4]`http://workshop2015.iwslt.org/`

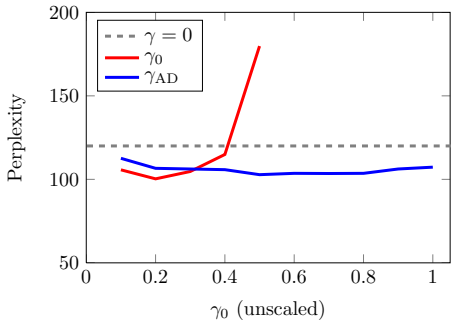


Figure 2: Perplexity with noising on Penn Treebank while varying the value of γ_0. Using discounting to scale γ_0 (yielding γ_{AD}) maintains gains for a range of values of noising probability, which is not true for the unscaled case.

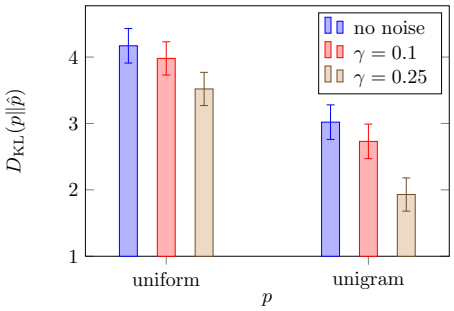


Figure 3: Mean KL-divergence over validation set between softmax distributions of noised and unnoised models and lower order distributions. Noised model distributions are closer to the uniform and unigram frequency distributions.

We train a two-layer LSTM encoder-decoder network (Sutskever et al., 2014; Cho et al., 2014) with 512 hidden units in each layer. The decoder uses an attention mechanism (Bahdanau et al., 2014) with the dot alignment function (Luong et al., 2015). The initial learning rate is 1.0 and we start halving the learning rate when the relative difference in perplexity on the validation set between two consecutive epochs is less than 1%. We follow training protocols as described in Sutskever et al. (2014): (a) LSTM parameters and word embeddings are initialized from a uniform distribution between $[-0.1, 0.1]$, (b) inputs are reversed, (c) batch size is set to 128, (d) gradient clipping is performed when the norm exceeds a threshold of 5. We set hidden unit dropout rate to 0.2 across all settings as suggested in Luong et al. (2015). We compare unigram, blank, and bigram Kneser-Ney noising. Noising rate γ is selected on the validation set.

Results are shown in Table 4. We observe performance gains for both blank noising and unigram noising, giving roughly $+0.7$ BLEU score on the test set. The proposed bigram Kneser-Ney noising scheme gives an additional performance boost of $+0.5$-0.7 on top of the blank noising and unigram noising models, yielding a total gain of $+1.4$ BLEU.

5 Discussion

5.1 Scaling γ via Discounting

We now examine whether discounting has the desired effect of noising subsequences according to their uncertainty. If we consider the discounting

$$\gamma_{\text{AD}}(x_1) = \gamma_0 \frac{N_{1+}(x_1, \bullet)}{c(x_1)}$$

we observe that the denominator $c(x_1)$ can dominate than the numerator $N_{1+}(x_1, \bullet)$. Common tokens are often noised infrequently when discounting is used to rescale the noising probability, while rare tokens are noised comparatively much more frequently, where in the extreme case when a token appears exactly once, we have $\gamma_{\text{AD}} = \gamma_0$. Due to word frequencies following a Zipfian power law distribution, however, common tokens constitute the majority of most texts, and thus discounting leads to significantly less noising.

We compare the performance of models trained with a fixed γ_0 versus a γ_0 rescaled using discounting. As shown in Figure 2, bigram discounting leads to gains in perplexity for a much broader range of γ_0. Thus the discounting ratio seems to effectively capture the "right" tokens to noise.

Noising	Bigrams	Trigrams
none (dropout only)	2881	381
blank noising	2760	372
unigram noising	2612	365

Table 5: Perplexity of last unigram for unseen bigrams and trigrams in Penn Treebank validation set. We compare noised and unnoised models with noising probabilities chosen such that models have near-identical perplexity on full validation set.

5.2 Noised versus Unnoised Models

Smoothed distributions In order to validate that data noising for RNN models has a similar effect to that of smoothing counts in n-gram models, we consider three models trained with unigram noising as described in Section 4.1 on the Penn Treebank corpus with $\gamma = 0$ (no noising), $\gamma = 0.1$, and $\gamma = 0.25$. Using the trained models, we measure the Kullback-Leibler divergence $D_{\mathrm{KL}}(p\|q) = \sum_i p_i \log(p_i/q_i)$ over the validation set between the predicted softmax distributions, $\hat{p}$, and the uniform distribution as well as the unigram frequency distribution. We then take the mean KL divergence over all tokens in the validation set.

Recall that in interpolation smoothing, a weighted combination of higher and lower order n-gram models is used. As seen in Figure 3, the softmax distributions of noised models are significantly closer to the lower order frequency distributions than unnoised models, in particular in the case of the unigram distribution, thus validating our analysis in Section 3.3.

Unseen n-grams Smoothing is most beneficial for increasing the probability of unobserved sequences. To measure whether noising has a similar effect, we consider bigrams and trigrams in the validation set that do not appear in the training set. For these unseen bigrams (15062 occurrences) and trigrams (43051 occurrences), we measure the perplexity for noised and unnoised models with near-identical perplexity on the full set. As expected, noising yields lower perplexity for these unseen instances.

6 Conclusion

In this work, we show that data noising is effective for regularizing neural network-based sequence models. By deriving a correspondence between noising and smoothing, we are able to adapt advanced smoothing methods for n-gram models to the neural network setting, thereby incorporating well-understood generative assumptions of language. Possible applications include exploring noising for improving performance in low resource settings, or examining how these techniques generalize to sequence modeling in other domains.

Acknowledgments

We thank Will Monroe for feedback on a draft of this paper, Anand Avati for help running experiments, and Jimmy Wu for computing support. We also thank the developers of Theano (Theano Development Team, 2016) and Tensorflow (Abadi et al., 2016). Some GPUs used in this work were donated by NVIDIA Corporation. ZX, SW, and JL were supported by an NDSEG Fellowship, NSERC PGS-D Fellowship, and Facebook Fellowship, respectively. This project was funded in part by DARPA MUSE award FA8750-15-C-0242 AFRL/RIKF.

References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Neural Information Processing Systems (NIPS)*, 2015.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. In *Journal Of Machine Learning Research*, 2003.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 1992.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Association for Computational Linguistics (ACL)*, 1996.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pp. 3061–3069, 2015.

Li Deng, Alex Acero, Mike Plumpe, and Xuedong Huang. Large-vocabulary speech recognition under adverse acoustic environments. In *ICSLP*, 2000.

Yarin Gal. A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*, 2015.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Association for Computatonal Linguistics (ACL)*, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

David Krueger and Roland Memisevic. Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*, 2015.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*, 2015.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Minh-Thang Luong and Christopher D Manning. Stanford neural machine translation systems for spoken language domains. In *Proceedings of the International Workshop on Spoken Language Translation*, 2015.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

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

Tomáš Mikolov. *Statistical language models based on neural networks*. PhD thesis, PhD thesis, Brno University of Technology. 2012.[PDF], 2012.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318. Association for Computational Linguistics, 2002.

Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, 2014.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

S. Wager, W. Fithian, S. I. Wang, and P. Liang. Altitude training: Strong bounds for single-layer dropout. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Stefan Wager, William Fithian, and Percy Liang. Data augmentation via levy processes. *arXiv preprint arXiv:1603.06340*, 2016.

Sida I Wang, Mengqiu Wang, Stefan Wager, Percy Liang, and Christopher D Manning. Feature noising for log-linear structured prediction. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

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

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.

A SKETCH OF NOISING ALGORITHM

We provide pseudocode of the noising algorithm corresponding to bigram Kneser-Ney smoothing for n-grams (In the case of sequence-to-sequence tasks, we estimate the count-based parameters separately for source and target). To simplify, we assume a batch size of one. The noising algorithm is applied to each data batch during training. No noising is applied at test time.

Algorithm 1 Bigram KN noising (Language modeling setting)

Require counts $c(x)$, number of distinct continuations $N_{1+}(x, \bullet)$, proposal distribution $q(x) \propto N_{1+}(\bullet, x)$
Inputs X, Y batch of unnoised data indices, scaling factor γ_0

procedure NOISEBGKN(X, Y) ▷ $X = (x_1, \ldots, x_t), Y = (x_2, \ldots, x_{t+1})$
  $\tilde{X}, \tilde{Y} \leftarrow X, Y$
  for $j = 1, \ldots, t$ **do**
    $\gamma \leftarrow \gamma_0 N_{1+}(x_j, \bullet)/c(x_j)$
    if $\sim \text{Bernoulli}(\gamma)$ **then**
      $\tilde{x}_j \sim \text{Categorical}(q)$ ▷ Updates $\tilde{X}$
      $\tilde{y}_j \sim \text{Categorical}(q)$
    end if
  end for
  return $\tilde{X}, \tilde{Y}$ ▷ Run training iteration with noised batch
end procedure

View publication stats