# Noisin: Unbiased Regularization for Recurrent Neural Networks

**Adji B. Dieng**[1] **Rajesh Ranganath**[2] **Jaan Altosaar**[3] **David M. Blei**[1]

## Abstract

Recurrent neural networks (RNNs) are powerful models of sequential data. They have been successfully used in domains such as text and speech. However, RNNs are susceptible to overfitting; regularization is important. In this paper we develop Noisin, a new method for regularizing RNNs. Noisin injects random noise into the hidden states of the RNN and then maximizes the corresponding marginal likelihood of the data. We show how Noisin applies to any RNN and we study many different types of noise. Noisin is unbiased—it preserves the underlying RNN on average. We characterize how Noisin regularizes its RNN both theoretically and empirically. On language modeling benchmarks, Noisin improves over dropout by as much as $12.2\%$ on the Penn Treebank and $9.4\%$ on the Wikitext-2 dataset. We also compared the state-of-the-art language model of Yang et al. 2017, both with and without Noisin. On the Penn Treebank, the method with Noisin more quickly reaches state-of-the-art performance.

## 1. Introduction

Recurrent neural networks (RNNs) are powerful models of sequential data (Robinson & Fallside, 1987; Werbos, 1988; Williams, 1989; Elman, 1990; Pearlmutter, 1995). RNNs have achieved state-of-the-art results on many tasks, including language modeling (Mikolov & Zweig, 2012; Yang et al., 2017), text generation (Graves, 2013), image generation (Gregor et al., 2015), speech recognition (Graves et al., 2013; Chiu et al., 2017), and machine translation (Sutskever et al., 2014; Wu et al., 2016).

The main idea behind an RNN is to posit a sequence of recursively defined *hidden states*, and then to model each observation conditional on its state. The key element of an RNN is its *transition function*. The transition function determines how each hidden state is a function of the previous observation and previous hidden state; it defines the underlying recursion. There are many flavors of RNNs—examples include the Elman Recurrent Neural Network (ERNN) (Elman, 1990), the Long-Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997), and the Gated Recurrent Unit (GRU) (Cho et al., 2014). Each flavor amounts to a different way of designing and parameterizing the transition function.

We fit an RNN by maximizing the likelihood of the observations with respect to its parameters, those of the transition function and of the observation likelihood. But RNNs are very flexible and they overfit; regularization is crucial. Researchers have explored many approaches to regularizing RNNs, such as Tikhonov regularization (Bishop, 1995), dropout and its variants (Srivastava et al., 2014; Zaremba et al., 2014; Gal & Ghahramani, 2016; Wan et al., 2013), and zoneout (Krueger et al., 2016). (See the related work section below for more discussion.)

In this paper, we develop Noisin, an effective new way to regularize an RNN. The idea is to inject random noise into its transition function and then to fit its parameters to maximize the corresponding marginal likelihood of the observations. We can easily apply Noisin to any flavor of RNN and we can use many types of noise.

Figure 1 demonstrates how an RNN can overfit and how Noisin can help. The plot involves a language modeling task where the RNN models a sequence of words. The horizontal axis is epochs of training; the vertical axis is perplexity, which is an assessment of model fitness (lower numbers are better). The figure shows how the model fits to both the training set and the validation set. As training proceeds, the vanilla RNN improves its fitness to the training set but performance on the validation set degrades—it overfits. The performance of the RNN with Noisin continues to improve in both the training set and the validation set.

Noisin regularizes the RNN by smoothing its loss, averaging over local neighborhoods of the transition function. Further, Noisin requires that the noise-injected transition function be *unbiased*. This means that, on average, it preserves the transition function of the original RNN.

[1]Columbia University [2]New York University [3]Princeton University. Correspondence to: Adji B. Dieng <abd2141@columbia.edu>.

*Proceedings of the $35^{th}$ International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

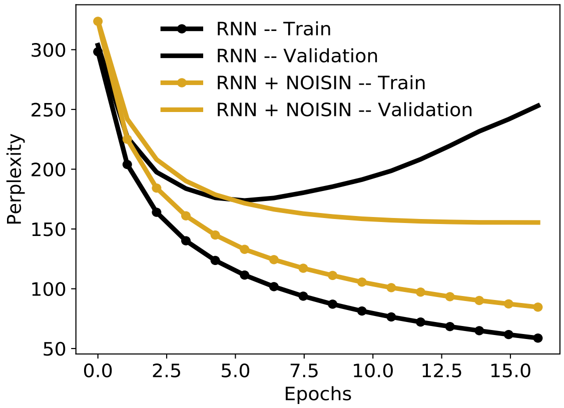

*Figure 1.* Training and validation perplexity for the deterministic RNN and the RNN regularized with Noisin. The settings were the same for both. We used additive Gaussian noise on an ERNN with sigmoid activations. We used one layer of 256 hidden units. The RNN overfits after only five epochs, and its training loss still decreases. This is not the case for the RNN regularized with Noisin.

With this requirement, we show that Noisin provides explicit regularization, i.e., it is equivalent to fitting the usual RNN loss plus a penalty function of its parameters. We can characterize the penalty as a function of the variance of the noise. Intuitively, it penalizes the components of the model that are sensitive to noise; this induces robustness to how future data may be different from the observations.

We examine Noisin with the LSTM and the LSTM with dropout, which we call the dropout-LSTM, and we explore several types of distributions. We study performance with two benchmark datasets on a language modeling task. Noisin improves over the LSTM by as much as 37.3% on the Penn Treebank dataset and 39.0% on the Wikitext-2 dataset; it improves over the dropout-LSTM by as much as 12.2% on the Penn Treebank and 9.4% on Wikitext-2.

**Related work.** Many techniques have been developed to address overfitting in RNNs. The most traditional regularization technique is weight decay ($L_1$ and $L_2$). However, Pascanu et al. (2013) showed that such simple regularizers prevent the RNNs from learning long-range dependencies.

Another technique for regularizing RNNs is to normalize the hidden states or the observations (Ioffe & Szegedy, 2015; Ba et al., 2016; Cooijmans et al., 2016). Though powerful, this class of approaches can be expensive.

Other types of regularization, including what we study in this paper, involve auxiliary noise variables. The most successful noise-based regularizer for neural networks is dropout (Srivastava et al., 2014; Wager et al., 2013; Noh et al., 2017). Dropout has been adapted to RNNs by only pruning their input and output matrices (Zaremba et al., 2014) or by putting judiciously chosen priors on all the weights and applying variational methods (Gal & Ghahramani, 2016). Still other noise-based regularization prunes the network by dropping updates to the hidden units of the RNN (Krueger et al., 2016; Semeniuta et al., 2016). More recently Merity et al. (2017) extended these techniques.

Involving noise variables in RNNs has been used in contexts other than regularization. For example Jim et al. (1996) analyze the impact of noise on convergence and long-term dependencies. Other work introduces auxiliary latent variables that enable RNNs to capture the high variability of complex sequential data such as music, audio, and text (Bayer & Osendorfer, 2014; Chung et al., 2015; Fraccaro et al., 2016; Goyal et al., 2017).

## 2. Recurrent Neural Networks

Consider a sequence of observations, $\boldsymbol{x}_{1:T} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$. An RNN factorizes its joint distribution according to the chain rule of probability,

$$p(\boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1}). \tag{1}$$

To capture dependencies, the RNN expresses each conditional probability as a function of a low-dimensional recurrent hidden state,

$$\boldsymbol{h}_t = f_W(\boldsymbol{x}_{t-1}, \boldsymbol{h}_{t-1}) \text{ and } p(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1}) = p(\boldsymbol{x}_t | \boldsymbol{h}_t).$$

The likelihood $p(\boldsymbol{x}_t|\boldsymbol{h}_t)$ can be of any form. We focus on the exponential family

$$p(\boldsymbol{x}_t|\boldsymbol{h}_t) = \nu(\boldsymbol{x}_t) \exp\left\{(V^\top \boldsymbol{h}_t)^\top \boldsymbol{x}_t - A(V^\top \boldsymbol{h}_t)\right\}, \tag{2}$$

where $\nu(\cdot)$ is the base measure, $V^\top \boldsymbol{h}_t$ is the natural parameter—a linear function of the hidden state $\boldsymbol{h}_t$—and $A(V^\top \boldsymbol{h}_t)$ is the log-normalizer. The matrix $V$ is called the *prediction* or *output* matrix of the RNN.

The hidden state $\boldsymbol{h}_t$ at time $t$ is a parametric function $f_W(\boldsymbol{h}_{t-1}, \boldsymbol{x}_{t-1})$ of the previous hidden state $\boldsymbol{h}_{t-1}$ and the previous observation $\boldsymbol{x}_{t-1}$; the parameters $W$ are shared across all time steps. The function $f_W$ is the transition function of the RNN, it defines a recurrence relation for the hidden states and renders $\boldsymbol{h}_t$ a function of all the past observations $\boldsymbol{x}_{1:t-1}$; these properties match the chain rule decomposition in Eq. 1.

The particular form of $f_W$ determines the RNN. Researchers have designed many flavors, including the LSTM and the GRU (Hochreiter & Schmidhuber, 1997; Cho et al., 2014).

*Table 1.* Expression for the log normalizer $A$ and its Hessian $\nabla^2 A$ for different likelihoods. Here $\sigma^2$ is the observation variance in the Gaussian case and $\eta = \exp(\boldsymbol{s})$ in the categorical case.

| Likelihood | $A(\boldsymbol{s})$ | $\nabla^2 A(\boldsymbol{s})$ |
|---|---|---|
| Bernoulli (Binary data) | $-\log(1-\sigma(\boldsymbol{s}))$ | $\sigma(\boldsymbol{s}) \cdot (1-\sigma(\boldsymbol{s}))$ |
| Gaussian (Real-Valued data) | $\frac{1}{2\sigma^2}\boldsymbol{s}^\top \boldsymbol{s}$ | $\frac{1}{\sigma^2}\boldsymbol{I}$ |
| Poisson (Count data) | $\exp(\boldsymbol{s})$ | $\exp(\boldsymbol{s})$ |
| Categorical (Categorical data) | $\text{logsumexp}(\eta)$ | $\frac{1}{\mathbb{1}^\top \eta}\text{diag}(\eta) - \frac{\eta\eta^\top}{(\mathbb{1}^\top \eta)^2}$ |

In this paper we will study the LSTM. However, the methods we develop can be applied to all types of RNNs.

**Long-Short Term Memory.** We now describe the LSTM, a variant of RNN that we study in Section 5. The LSTM is built from the simpler ERNN (Elman, 1990). In an ERNN, the transition function is

$$f_W(\boldsymbol{x}_{t-1}, \boldsymbol{h}_{t-1}) = s(W_x^\top \boldsymbol{x}_{t-1} + W_h^\top \boldsymbol{h}_{t-1}),$$

where we dropped an intercept term to avoid cluttered notation. Here, $W_h$ is called the *recurrent weight matrix* and $W_x$ is called the *embedding matrix* or *input matrix*. The function $s(\cdot)$ is called an *activation* or *squashing* function, which stabilizes the transition dynamics by bounding the hidden state. Typical choices for the squashing function include the sigmoid and the hyperbolic tangent.

The LSTM was designed to avoid optimization issues, such as vanishing (or exploding) gradients. Its transition function composes four ERNNs, three with sigmoid activations and one with a tanh activation:

$$f_t = \sigma(W_{x1}^\top \boldsymbol{x}_{t-1} + W_{h1}^\top \boldsymbol{h}_{t-1}) \quad (3)$$
$$i_t = \sigma(W_{x2}^\top \boldsymbol{x}_{t-1} + W_{h2}^\top \boldsymbol{h}_{t-1}) \quad (4)$$
$$o_t = \sigma(W_{x4}^\top \boldsymbol{x}_{t-1} + W_{h4}^\top \boldsymbol{h}_{t-1}) \quad (5)$$
$$\boldsymbol{c}_t = f_t \odot \boldsymbol{c}_{t-1} + i_t \odot \tanh(W_{x3}^\top \boldsymbol{x}_{t-1} + W_{h3}^\top \boldsymbol{h}_{t-1}) \quad (6)$$
$$\boldsymbol{h}_t = o_t \odot \tanh(\boldsymbol{c}_t). \quad (7)$$

The idea is that the memory cell $\boldsymbol{c}_t$ captures long-term dependencies (Hochreiter & Schmidhuber, 1997).

However, LSTMs have a high model complexity and, consequently, they easily memorize data. Regularization is crucial. In the next section, we develop a new regularization method for RNNs called Noisin.

## 3. Noise-Injected RNNs

Noisin is built from noise-injected recurrent neural network (RNN)s. These are RNNs whose hidden states are computed using auxiliary noise variables. There are several advantages to injecting noise into the hidden states of RNNs. For example it prevents the dimensions of the hidden states from co-adapting and forces individual units to capture useful features.

We define noise-injected RNNs as any RNN following the generative process

$$\boldsymbol{\epsilon}_{1:T} \sim \varphi(\cdot; \mu, \gamma) \quad (8)$$
$$\boldsymbol{z}_t = g_W(\boldsymbol{x}_{t-1}, \boldsymbol{z}_{t-1}, \boldsymbol{\epsilon}_t) \quad (9)$$
$$p(\boldsymbol{x}_t \,|\, \boldsymbol{x}_{1:t-1}) = p(\boldsymbol{x}_t \,|\, \boldsymbol{z}_t), \quad (10)$$

where the likelihood $p(\boldsymbol{x}_t \,|\, \boldsymbol{z}_t)$ is an exponential family as in Eq. 2. The noise variables $\boldsymbol{\epsilon}_{1:T}$ are drawn from a distribution $\varphi(\cdot; \mu, \gamma)$ with mean $\mu$ and scale $\gamma$. For example, $\varphi(\cdot; \mu, \gamma)$ can be a zero-mean Gaussian with variance $\gamma^2$. We will study many types of noise distributions.

The noisy hidden state $\boldsymbol{z}_t$ is a parametric function $g_W$ of the previous observation $\boldsymbol{x}_{t-1}$, the previous noisy hidden state $\boldsymbol{z}_{t-1}$, and the noise $\boldsymbol{\epsilon}_t$. Therefore conditional on the noise $\boldsymbol{\epsilon}_{1:T}$, the transition function $g_W$ defines a recurrence relation on $\boldsymbol{z}_{1:T}$.

The function $g_W$ determines the noise-injected RNN. In this paper, we propose functions $g_W$ that meet the criterion described below.

**Unbiased noise injection.** Injecting noise at each time step limits the amount of information carried by hidden states. In limiting their capacity, noise injection is some form of regularization. In Section 4 we show that noise injection under exponential family likelihoods corresponds to explicit regularization under some *unbiasedness* condition.

We define two flavors of unbiasedness: *strong unbiasedness* and *weak unbiasedness*. Let $\boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t})$ denote the unrolled recurrence at time $t$; it is a random variable via the noise $\boldsymbol{\epsilon}_{1:t}$. Under the strong unbiasedness condition, the transition function $g_W$ must satisfy the relationship

$$\mathbb{E}_{p(\boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t}) \,|\, \boldsymbol{z}_{t-1})}\left[\boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t})\right] = \boldsymbol{h}_t \quad (11)$$

where $\boldsymbol{h}_t$ is the hidden state of the underlying RNN. This is satisfied by injecting the noise at the last layer of the RNN. Weak unbiasedness imposes a looser constraint. Under weak unbiasedness, $g_W$ must satisfy

$$\mathbb{E}_{p(\boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t}) \,|\, \boldsymbol{z}_{t-1})}\left[\boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t})\right] = f_W(\boldsymbol{x}_{t-1}, \boldsymbol{z}_{t-1}) \quad (12)$$

where $f_W$ is the transition function of the underlying RNN. What weak unbiasedness means is that the noise should be injected in such a way that driving the noise to zero leads to the original RNN. Two possible choices for $g_W$ that meet this condition are the following

$$g_W(\boldsymbol{x}_{t-1}, \boldsymbol{z}_{t-1}, \boldsymbol{\epsilon}_t) = f_W(\boldsymbol{x}_{t-1}, \boldsymbol{z}_{t-1}) + \boldsymbol{\epsilon}_t \tag{13}$$

$$g_W(\boldsymbol{x}_{t-1}, \boldsymbol{z}_{t-1}, \boldsymbol{\epsilon}_t) = f_W(\boldsymbol{x}_{t-1}, \boldsymbol{z}_{t-1}) \odot \boldsymbol{\epsilon}_t. \tag{14}$$

In Eq. 13 the noise has mean zero whereas in Eq. 14 it has mean one. These choices of $g_W$ correspond to additive noise and multiplicative noise respectively. Note $f_W$ can be any RNN including the RNN with dropout or the stochastic RNNs (Bayer & Osendorfer, 2014; Chung et al., 2015; Fraccaro et al., 2016; Goyal et al., 2017). For example to implement unbiased noise injection with multiplicative noise for the Long-Short Term Memory (LSTM) the only change from the original LSTM is to replace Eq. 7 with

$$\boldsymbol{z}_t = o_t \odot \tanh(\boldsymbol{c}_t) \odot \boldsymbol{\epsilon}_t.$$

Such noise-injected hidden states can be stacked to build a multi-layered noise-injected LSTM that meet the weak unbiasedness condition.

**Dropout.** We now consider dropout from the perspective of unbiasedness. Consider the LSTM as described in Section 2. Applying dropout to it corresponds to injecting Bernoulli-distributed noise as follows

$$\begin{aligned} f_t &= \sigma(W_{x1}^\top \boldsymbol{x}_{t-1} \odot \boldsymbol{\epsilon}_t^{xf} + W_{h1}^\top \boldsymbol{h}_{t-1} \odot \boldsymbol{\epsilon}_t^{hf}) \\ i_t &= \sigma(W_{x2}^\top \boldsymbol{x}_{t-1} \odot \boldsymbol{\epsilon}_t^{xi} + W_{h2}^\top \boldsymbol{h}_{t-1} \odot \boldsymbol{\epsilon}_t^{hi}) \\ o_t &= \sigma(W_{x4}^\top \boldsymbol{x}_{t-1} \odot \boldsymbol{\epsilon}_t^{xo} + W_{h4}^\top \boldsymbol{h}_{t-1} \odot \boldsymbol{\epsilon}_t^{ho}) \\ \boldsymbol{c}_t &= f_t \odot \boldsymbol{c}_{t-1} + \\ i_t &\odot \tanh(W_{x3}^\top \boldsymbol{x}_{t-1} \odot \boldsymbol{\epsilon}_t^{xc} + W_{h3}^\top \boldsymbol{h}_{t-1} \odot \boldsymbol{\epsilon}_t^{hc}) \\ \boldsymbol{z}_t^{dropout} &= o_t \odot \tanh(\boldsymbol{c}_t). \end{aligned}$$

This general form of dropout encapsulates existing dropout variants. For example when the noise variables $\boldsymbol{\epsilon}_t^{hf}, \boldsymbol{\epsilon}_t^{hi}, \boldsymbol{\epsilon}_t^{ho}, \boldsymbol{\epsilon}_t^{hc}$ are set to one we recover the variant of dropout in Zaremba et al. (2014).

Because of the nonlinearities dropout does not meet the unbiasedness desideratum Eq. 12 where $\boldsymbol{h}_t$ is the hidden state of the LSTM as described in Section 2. Here at each time step $t$, $\boldsymbol{\epsilon}_t$ denotes the set of noise variables $\boldsymbol{\epsilon}_t^{xf}, \boldsymbol{\epsilon}_t^{xi}, \boldsymbol{\epsilon}_t^{xo}, \boldsymbol{\epsilon}_t^{xc}$ and $\boldsymbol{\epsilon}_t^{hf}, \boldsymbol{\epsilon}_t^{hi}, \boldsymbol{\epsilon}_t^{ho}, \boldsymbol{\epsilon}_t^{hc}$.

Dropout is therefore biased and does not preserve the underlying RNN. However, dropout has been widely successfully used in practice and has many nice properties. For example it regularizes by acting like an ensemble method (Goodfellow et al., 2016). We study the dropout-LSTM in Section 5 as a variant of RNN that can benefit from the method Noisin proposed in this paper.

**Algorithm 1** Noisin with multiplicative noise.

```
Input: Data $\boldsymbol{x}_{1:T}$, initial hidden state $\boldsymbol{z}_0$, noise distribution $\varphi(\cdot; 1, \gamma)$, and learning rate $\rho$.
Output: learned parameters $W^*$ and $V^*$.
Initialize parameters $W$ and $V$
for iteration iter $= 1, 2, \ldots,$ do
  for time step $t = 1, \ldots, T$ do
    Sample noise $\boldsymbol{\epsilon}_t \sim \varphi(\boldsymbol{\epsilon}_t; 1, \gamma)$
    Compute state $\boldsymbol{z}_t = f_W(\boldsymbol{z}_{t-1}, \boldsymbol{x}_{t-1}) \odot \boldsymbol{\epsilon}_t$
  end for
  Compute loss $\widetilde{\mathcal{L}}$ as in Eq. 17
  Update $W :\leftarrow W - \rho \cdot \nabla_W \widetilde{\mathcal{L}}$
  Update $V :\leftarrow V - \rho \cdot \nabla_V \widetilde{\mathcal{L}}$
  Change learning rate $\rho$ according to some schedule.
end for
```

**Unbiased noise-injection with Noisin.** Deterministic RNNs are learned using truncated backpropagation through time with the maximum likelihood objective—the log likelihood of the data. Backpropagation through time builds gradients by *unrolling* the RNN into a feed-forward neural network and applies backpropagation (Rumelhart et al., 1988). The RNN is then optimized using gradient descent or stochastic gradient descent (Robbins & Monro, 1951).

Noisin applies the same procedure to the expected log-likelihood under the injected noise,

$$\mathcal{L} = E_{p(\boldsymbol{\epsilon}_{1:T})}\left[\log p(\boldsymbol{x}_{1:T} | \boldsymbol{z}_{1:T}(\boldsymbol{\epsilon}_{1:T}))\right]. \tag{15}$$

In more detail this is

$$\mathcal{L} = \sum_{t=1}^{T} E_{p(\boldsymbol{\epsilon}_{1:t})}\Big[\log p(\boldsymbol{x}_t | \boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t}))\Big] \tag{16}$$

Notice this objective is a Jensen bound on the marginal log-likelihood of the data,

$$\mathcal{L} \leq \log E_{p(\boldsymbol{\epsilon}_{1:T})}\left[p(\boldsymbol{x}_{1:T} | \boldsymbol{z}_{1:T}(\boldsymbol{\epsilon}_{1:T}))\right] = \log p(\boldsymbol{x}_{1:T}).$$

The expectations in the objective of Eq. 16 are intractable due to the nonlinearities in the model and the form of the noise distribution. We approximate the objective using Monte Carlo;

$$\widehat{\mathcal{L}} = \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{T} \Big[\log p(\boldsymbol{x}_t | \boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t}^{(k)}))\Big].$$

When using one sample ($K = 1$), the training procedure is just as easy as for the underlying RNN. The loss in this case, under the exponential family likelihood, becomes

$$\widetilde{\mathcal{L}} = -\sum_{t=1}^{T} \left[(V^\top \boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t}))^\top \boldsymbol{x}_t - A(V^\top \boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t}))\right] + c, \tag{17}$$

where $c = -\sum_{t=1}^{T} \log \nu(\boldsymbol{x}_t)$ is a constant that does not depend on the parameters. Algorithm 1 summarizes the procedure for multiplicative noise. The only change from traditional RNN training is when updating the hidden state in lines 4 and 5.

**Controling the noise level.** Noisin is amenable to any RNN and any noise distribution. As with all regularization techniques, Noisin comes with a free parameter that determines the amount of regularization: the spread $\gamma$ of the noise.

Certain noise distributions have bounded variance; for example the Bernoulli and the Beta distributions. This limits the amount of regularization one can afford. To circumvent this bounded variance issue, we rescale the noise to have unbounded variance. Table 2 shows the expression of the variance of the original noise and its scaled version for several distributions. It is the scaled noise that is used in Noisin.

## 4. Unbiased Regularization for RNNs

In Section 3, we introduced the concept of unbiasedness in the context of RNNs as a desideratum for noise injection to preserve the underlying RNN. In this section we prove unbiasedness leads to an explicit regularizer that forces the hidden states to be robust to noise.

### 4.1. Unbiased noise injection is explicit regularization

A valid regularizer is one that adds a nonnegative term to the risk. This section shows that unbiased noise injection with exponential family likelihoods leads to valid regularizers.

Consider the loss in Eq. 17 for an exponential family likelihood. The exponential family provides a general notation for the types of data encountered in practice: binary, count, real-valued, and categorical. Table 1 shows the expression of $A$ for these types of data. The log normalizer $A(V^\top \boldsymbol{z}_t)$ has many useful properties. For example it is convex and infinitely differentiable.

Assume without loss of generality that we observe one sequence $\boldsymbol{x}_{1:T}$. Consider the empirical risk function for the noise-injected RNN. It is defined as

$$\mathcal{R} = -\sum_{t=1}^{T} E_{p(\boldsymbol{\epsilon}_{1:t})} \left\{ (V^\top \boldsymbol{z}_t)^\top \boldsymbol{x}_t - A(V^\top \boldsymbol{z}_t) \right\} + c.$$

With little algebra we can decompose this risk into the sum of two terms

$$\mathcal{R} = \mathcal{R}(\text{det}) + \sum_{t=1}^{T} E_{p(\boldsymbol{\epsilon}_{1:t})} \left\{ \mathcal{E}_t \right\} \tag{18}$$

where $\mathcal{R}(\text{det})$ is the empirical risk for the underlying RNN and $\mathcal{E}_t$ is

$$\mathcal{E}_t = A(V^\top \boldsymbol{z}_t) - A(V^\top \boldsymbol{h}_t) - \left(V^\top \boldsymbol{z}_t - V^\top \boldsymbol{h}_t\right)^\top \boldsymbol{x}_t.$$

Because the second term in Eq. 18 is not always guaranteed to be non-negative, noise-injection is not explicit regularization in general. However, under the strong unbiasedness condition, this term corresponds to a valid regularization term and simplifies to

$$\mathcal{R}\text{eg} = \frac{1}{2} \sum_{t=1}^{T} \text{tr} \left\{ E_{p(\boldsymbol{\epsilon}_{1:t})} \text{Cov}(B^\top \boldsymbol{z}_t \,|\, \boldsymbol{z}_{t-1}(\boldsymbol{\epsilon}_{1:t-1})) \right\},$$

where the matrix $B = V \sqrt{\nabla^2 A(V^\top \boldsymbol{h}_t)}$ is the prediction matrix of the underlying RNN rescaled by the square root of $\nabla^2 A(V^\top \boldsymbol{h}_t)$—the Hessian of the log-normalizer of the likelihood. This Hessian is also the Fisher information matrix of the RNN. We provide a detailed proof in Section 7.

Noisin requires that we minimize the objective of the underlying RNN while also minimizing $\mathcal{R}$eg. Minimizing $\mathcal{R}$eg induces robustness—it is equivalent to penalizing hidden units that are too sensitive to noise.

### 4.2. Connections

In this section, we intuit that Noisin has ties to ensemble methods and empirical Bayes.

**The ensemble method perspective.** Noisin can be interpreted as an ensemble method. The objective in Eq. 16 corresponds to averaging the predictions of infinitely many RNNs at each time step in the sequence. This is known as an ensemble method and has a regularization effect (Poggio et al., 2002). However ensemble methods are costly as they require training all the sub-models in the ensemble. With Noisin, at each time step in the sequence, one of the infinitely many RNNs is trained and because of parameter sharing, the RNN being trained at the next time step will use better settings of the weights. This makes training the whole model efficient. (See Algorithm 1.)

**The empirical Bayes perspective.** Consider a noise-injected RNN. We write its joint distribution as

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{x}_t | \boldsymbol{z}_t; V) p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1}, \boldsymbol{x}_{t-1}; W)$$

Here $p(\boldsymbol{x}_t | \boldsymbol{z}_t; V)$ denotes the likelihood and $p(\boldsymbol{z}_t | \boldsymbol{z}_{t-1}, \boldsymbol{x}_{t-1}; W)$ is the prior over the noisy hidden states; it is parameterized by the weights $W$. From the perspective of Bayesian inference this is an unknown prior. When we optimize the objective in Eq. 16, we are learning the weights $W$. This is equivalent to learning the prior over the noisy hidden states and is known as empirical Bayes

*Table 2.* Expression for the noise distributions and their scaled version used in this paper. Here $\gamma$ is the noise spread. It determines the amount of regularization. For example it is the standard deviation for Gaussian noise and the scale parameter for Gamma noise. The constant $\delta = 0.5772$ is the Euler-Mascheroni constant

| Standard Noise $\eta$ | $E(\eta)$ | $Var(\eta)$ | Scaled Noise $\boldsymbol{\epsilon}$ | $E(\boldsymbol{\epsilon})$ | $Var(\boldsymbol{\epsilon})$ |
|---|---|---|---|---|---|
| $\mathcal{N}(0, \gamma)$ | $0$ | $\gamma^2$ | $\eta$ | $0$ | $\gamma^2$ |
| Bernoulli$(\gamma)$ | $\gamma$ | $\gamma(1-\gamma)$ | $\frac{\eta}{\gamma}$ | $1$ | $\frac{1-\gamma}{\gamma}$ |
| Gamma$(\alpha, \gamma)$ | $\alpha\gamma$ | $\alpha\gamma^2$ | $\frac{\eta-\alpha\gamma}{\sqrt{\alpha}}$ | $0$ | $\gamma^2$ |
| Gumbel$(0, \gamma)$ | $\delta\gamma$ | $\frac{\pi^2\gamma^2}{6}$ | $\frac{\sqrt{6}(\eta-\delta\gamma)}{\pi}$ | $0$ | $\gamma^2$ |
| Laplace$(0, \gamma)$ | $0$ | $2\gamma^2$ | $\frac{\eta}{\sqrt{2}}$ | $0$ | $\gamma^2$ |
| Logistic$(0, \gamma)$ | $0$ | $\frac{\pi^2\gamma^2}{3}$ | $\frac{\sqrt{3}\eta}{\pi}$ | $0$ | $\gamma^2$ |
| Beta$(\alpha, \gamma)$ | $\frac{\alpha}{\alpha+\gamma}$ | $\frac{\alpha\gamma}{(\alpha+\gamma)^2(\alpha+\gamma+1)}$ | $(\alpha+\gamma)\sqrt{\frac{\alpha+\gamma+1}{\alpha}}(\eta - \frac{\alpha}{\alpha+\gamma})$ | $0$ | $\gamma$ |
| Chi-Square$(\gamma)$ | $\gamma$ | $2\gamma$ | $\frac{\eta-\gamma}{\sqrt{2}}$ | $0$ | $\gamma$ |

(Robbins, 1964). It consists in getting point estimates of prior parameters in a hierarchical model and using those point estimates to define the prior.

## 5. Empirical Study

We presented Noisin, a method that relies on unbiased noise injection to regularize any RNN. Noisin is simple and can be integrated with any existing RNN-based model. In this section, we focus on applying Noisin to the LSTM and the dropout-LSTM. We use language modeling as a testbed. Regularization is crucial in language modeling because the input and prediction matrices scale linearly with the size of the vocabulary. This results in networks with very high capacity.

We used Noisin under two noise regimes: additive noise and multiplicative noise. We found that additive noise uniformly performs worse than multiplicative noise for the LSTM. We therefore report results only on multiplicative noise.

We used Noisin with several noise distributions: Gaussian, Logistic, Laplace, Gamma, Bernoulli, Gumbel, Beta, and $\chi$-Square. We found that overall the only property that matters with these distributions is the variance. The variance determines the amount of regularization for Noisin. It is the parameter $\gamma$ in Algorithm 1. We outlined in Section 4 how to set the noise level for a given distribution so as to benefit from unbounded variance.

We also found that these distributions, when used with Noisin on the LSTM perform better than the dropout LSTM on the Penn Treebank.

Another interesting finding is that Noisin when applied to the dropout-LSTM performs better than the original dropout-LSTM.

Next we describe the two benchmark datasets used: Penn Treebank and Wikitext-2. We then provide details on the experimental settings for reproducibility. We finally present the results in Table 3 and Table 4.

**Penn Treebank.** The Penn Treebank portion of the Wall Street Journal (Marcus et al., 1993) is a long standing benchmark dataset for language modeling. We use the standard split, where sections 0 to 20 ($930K$ tokens) are used for training, sections 21 to 22 ($74K$ tokens) for validation, and sections 23 to 24 ($82K$ tokens) for testing (Mikolov et al., 2010). We use a vocabulary of size $10K$ that includes the special token *unk* for rare words and the end of sentence indicator *eos*.

**Wikitext-2.** The Wikitext-2 dataset (Merity et al., 2016) has been recently introduced as an alternative to the Penn Treebank dataset. It is sourced from Wikipedia articles and is approximately twice the size of the Penn Treebank dataset. We use a vocabulary size of $30K$ and no further preprocessing steps.

**Experimental settings.** To assess the capabilities of Noisin as a regularizer on its own, we used the basic settings for RNN training (Zaremba et al., 2014). We did not use weight decay or pointers (Merity et al., 2016).

We considered two settings in our experiments: a medium-sized network and a large network. The medium-sized network has 2 layers with 650 hidden units each. This results in a model complexity of 13 million parameters. The large network has 2 layers with 1500 hidden units each. This leads to a model complexity of 51 million parameters.

For each setting, we set the dimension of the word embeddings to match the number of hidden units in each layer. Following initialization guidelines in the literature, we initialize all embedding weights uniformly in the interval $[-0.1, 0.1]$. All other weights were initialized uniformly between $[-\frac{1}{\sqrt{H}}, \frac{1}{\sqrt{H}}]$ where $H$ is the number of hidden units in a layer. All the biases were initialized to $0$. We fixed the seed to $1111$ for reproducibility.

*Table 3.* Noisin improves the performance of the LSTM and the dropout-LSTM by as much as 12% on the Penn Treebank dataset. This table shows word-level perplexity scores on the medium and large settings for both the validation (or dev) and the test set.

| | Medium | | | Large | | | | Medium | | | Large | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | $\gamma$ | Dev | Test | $\gamma$ | Dev | Test | Method | $\gamma$ | Dev | Test | $\gamma$ | Dev | Test |
| None | −− | 115 | 109 | −− | 123 | 123 | Dropout (D) | −− | 80.2 | 77.0 | −− | 78.6 | 75.3 |
| Gaussian | 1.10 | 76.2 | 71.8 | 1.37 | 73.2 | 69.1 | D + Gaussian | 0.53 | 73.4 | 70.4 | 0.92 | **70.0** | **66.1** |
| Logistic | 1.06 | 76.4 | 72.3 | 1.39 | 73.6 | 69.3 | D + Logistic | 0.53 | 73.0 | 69.9 | 0.84 | 69.8 | 66.4 |
| Laplace | 1.06 | 76.6 | 72.4 | 1.39 | 73.7 | 69.4 | D + Laplace | 0.53 | 73.1 | 70.0 | 0.92 | 69.9 | 66.6 |
| Gamma | 1.06 | 78.2 | 74.5 | 1.39 | 73.6 | 69.5 | D + Gamma | 0.38 | 73.5 | 70.3 | 0.92 | 71.1 | 68.2 |
| Bernoulli | 0.41 | **75.7** | **71.4** | 0.33 | **72.8** | **68.3** | D + Bernoulli | 0.80 | 73.3 | 70.1 | 0.50 | **70.0** | **66.1** |
| Gumbel | 1.06 | 76.2 | 72.7 | 1.39 | 73.5 | 69.5 | D + Gumbel | 0.46 | 74.5 | 71.2 | 0.92 | 70.2 | 67.1 |
| Beta | 1.07 | 76.0 | 71.4 | 1.50 | 74.4 | 70.2 | D + Beta | 0.20 | **73.0** | **69.2** | 0.70 | 70.0 | 66.2 |
| Chi | 1.50 | 84.5 | 80.7 | 1.20 | 79.2 | 75.5 | D + Chi | 0.29 | 76.1 | 72.8 | 0.82 | 73.0 | 70.0 |

*Table 4.* Noisin improves the performance of the LSTM and the dropout-LSTM by as much as 9% on the Wikitext-2 dataset. This table shows word-level perplexity scores on the medium and large settings for both the validation (or dev) and the test set. D is short for dropout. $D +$ Distribution refers to Noisin applied to the dropout-LSTM with the specified distribution.

| | Medium | | | Large | | | | Medium | | | Large | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | $\gamma$ | Dev | Test | $\gamma$ | Dev | Test | Method | $\gamma$ | Dev | Test | $\gamma$ | Dev | Test |
| None | −− | 141 | 136 | −− | 176 | 140 | Dropout (D) | −− | 88.7 | 84.8 | −− | 95.0 | 91.0 |
| Gaussian | 1.00 | 92.7 | 87.8 | 1.37 | 87.7 | 83.4 | D + Gaussian | 0.50 | 86.3 | 82.3 | 0.69 | 81.4 | 77.7 |
| Logistic | 1.00 | 93.2 | 88.4 | 1.28 | 88.1 | 83.5 | D + Logistic | 0.40 | 86.4 | 82.5 | 0.77 | 81.6 | 78.1 |
| Laplace | 1.00 | 95.3 | 89.8 | 1.28 | 88.0 | 83.4 | D + Laplace | 0.40 | **85.6** | **82.1** | 0.61 | 83.2 | 79.1 |
| Gamma | 0.72 | 97.6 | 92.9 | 1.39 | 89.2 | 84.5 | D + Gamma | 0.30 | 86.5 | 82.4 | 0.61 | 85.5 | 81.3 |
| Bernoulli | 0.54 | 91.2 | 86.6 | 0.41 | 86.9 | 83.0 | D + Bernoulli | 0.50 | 100.6 | 94.4 | 0.64 | **80.8** | **76.8** |
| Gumbel | 1.00 | 95.4 | 90.9 | 1.28 | 88.7 | 84.0 | D + Gumbel | 0.30 | 86.4 | 82.4 | 0.53 | 83.7 | 80.1 |
| Beta | 0.80 | **91.1** | **87.2** | 1.50 | **86.9** | **82.9** | D + Beta | 0.10 | 86.2 | 82.3 | 0.60 | 81.5 | 77.9 |
| Chi | 0.20 | 111 | 105 | 1.50 | 99.0 | 92.9 | D + Chi | 0.20 | 92.0 | 87.4 | 0.29 | 87.1 | 82.8 |

*Table 5.* When applied to the model in (Yang et al., 2017), Noisin achieves the same state-of-the-art perplexity on the Penn Treebank after only 400 epochs (vs 1000 epochs). $^*$Multiplicative gamma-distributed noise with shape 2 and scale 0.4.

| Model | # Parameters | Dev | Test |
|---|---|---|---|
| (Zaremba et al., 2014) - LSTM | 20 M | 86.2 | 82.7 |
| (Gal & Ghahramani, 2016) - Variational LSTM (MC) | 20 M | – | 78.6 |
| (Merity et al., 2016) - Pointer Sentinel-LSTM | 21 M | 72.4 | 70.9 |
| (Grave et al., 2016) - LSTM + continuous cache pointer | – | – | 72.1 |
| (Inan et al., 2016) - Tied Variational LSTM + augmented loss | 24 M | 75.7 | 73.2 |
| (Zilly et al., 2016)- Variational RHN | 23 M | 67.9 | 65.4 |
| (Melis et al., 2017) - 2-layer skip connection LSTM | 24 M | 60.9 | 58.3 |
| (Merity et al., 2017) - AWD-LSTM + continuous cache pointer | 24 M | 53.9 | 52.8 |
| (Krause et al., 2017) - AWD-LSTM + dynamic evaluation | 24 M | 51.6 | 51.1 |
| (Yang et al., 2017) - AWD-LSTM-MoS + dynamic evaluation | 22 M | **48.3** | **47.7** |
| (This paper) - AWD-LSTM-MoS + Noisin$^*$ + dynamic evaluation | 22 M | **48.4** | **47.6** |

We train the models using truncated backpropagation through time with average stochastic gradient descent (Polyak & Juditsky, 1992) for a maximum of 200 epochs. The LSTM was unrolled for 35 steps. We used a batch size of 80 for both datasets. To avoid the problem of exploding gradients we clip the gradients to a maximum norm of $0.25$. We used an initial learning rate of $30$ for all experiments. This is divided by a factor of $1.2$ if the perplexity on the validation set deteriorates.

For the dropout-LSTM, the values used for dropout on the input, recurrent, and output layers were $0.5, 0.4, 0.5$ respectively.

The models were implemented in PyTorch. The source code is available upon request.

**Results on the Penn Treebank.** The results on the Penn Treebank are illustrated in Table 3. The best results for the non-regularized LSTM correspond to a small network. This is because larger networks overfit and require regularization. In general Noisin improves any given RNN including dropout-LSTM. For example Noisin with multiplicative Bernoulli noise performs better than dropout RNN for both medium and large settings. Noisin improves the performance of the dropout-LSTM by as much as $12.2\%$ on this dataset.

**Results on the Wikitext-2 dataset.** Results on the Wikitext-2 dataset are presented in Table 4. We observe the same trend as for the Penn Treebank dataset: Noisin improves the underlying LSTM and dropout-LSTM. For the dropout-LSTM, it improves its generalization capabilities by as much as $9\%$ on this dataset.

## 6. Discussion

We proposed Noisin, a simple method for regularizing RNNs. Noisin injects noise into the hidden states such that the underlying RNN is preserved. Noisin maximizes a lower bound of the log marginal likelihood of the data—the expected log-likelihood under the injected noise. We showed that Noisin is an explicit regularizer that imposes a robustness constraint on the hidden units of the RNN. On a language modeling benchmark Noisin improves the generalization capabilities of both the LSTM and the dropout-LSTM.

## 7. Detailed Derivations

We derive in full detail the risk of Noisin and show that it can be written as the sum of the risk of the original RNN and a regularization term.

Assume without loss of generality that we observe one sequence $\boldsymbol{x}_{1:T}$. The risk of a noise-injected RNN is

$$\mathcal{R} = -\sum_{t=1}^{T} E_{p(\boldsymbol{\epsilon}_{1:t})} \log p(\boldsymbol{x}_t | \boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t})).$$

Expand this in more detail and write $\boldsymbol{z}_t$ in lieu of $\boldsymbol{z}_t(\boldsymbol{\epsilon}_{1:t})$ to avoid cluttering of notation. Then

$$\mathcal{R} = -\sum_{t=1}^{T} \left\{ \log \nu(\boldsymbol{x}_t) - E_{p(\boldsymbol{\epsilon}_{1:t})} \left[ \boldsymbol{z}_t^\top V \boldsymbol{x}_t - A(V^\top \boldsymbol{z}_t) \right] \right\}.$$

The risk for the underlying RNN—$\mathcal{R}$(det)—is similar when we replace $\boldsymbol{z}_t$ with $\boldsymbol{h}_t$,

$$\mathcal{R}(\text{det}) = -\sum_{t=1}^{T} \left\{ \log \nu(\boldsymbol{x}_t) - \left[ \boldsymbol{h}_t^\top V \boldsymbol{x}_t - A(V^\top \boldsymbol{h}_t) \right] \right\}.$$

Therefore we can express the risk of Noisin as a function of the risk of the underlying RNN,

$$\mathcal{R} = \mathcal{R}(\text{det}) + \sum_{t=1}^{T} E_{p(\boldsymbol{\epsilon}_{1:t-1})} \left[ E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})} \left( \mathcal{E}_1 \right) \right]$$

$$\mathcal{E}_1 = A(V^\top \boldsymbol{z}_t) - A(V^\top \boldsymbol{h}_t) - \left( V^\top \boldsymbol{z}_t - V^\top \boldsymbol{h}_t \right)^\top \boldsymbol{x}_t.$$

Under the strong unbiasedness condition,

$$E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})} \left[ \mathcal{E}_1 \right) = E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})} \left[ A(V^\top \boldsymbol{z}_t) - A(V^\top \boldsymbol{h}_t) \right].$$

Using the convexity property of the log-normalizer of exponential families and Jensen's inequality,

$$E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})} \left( \mathcal{E}_1 \right) \geq A(V^\top E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})}(\boldsymbol{z}_t)) - A(V^\top \boldsymbol{h}_t).$$

Using the strong unbiasedness condition a second time we conclude $E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})} \left( \mathcal{E}_1 \right) \geq 0$. Therefore

$$\mathcal{R}\text{eg} = \sum_{t=1}^{T} E_{p(\boldsymbol{\epsilon}_{1:t-1})} \left[ E_{p(\boldsymbol{\epsilon}_t \,|\, \boldsymbol{\epsilon}_{1:t-1})} \left( \mathcal{E}_1 \right) \right] \geq 0$$

is a valid regularizer. A second-order Taylor expansion of $A(V^\top \boldsymbol{z}_t)$ around $A(V^\top \boldsymbol{h}_t)$ and the strong unbiasedness condition yield

$$\mathcal{R}\text{eg} = \frac{1}{2} \sum_{t=1}^{T} \text{tr} \left\{ E_{p(\boldsymbol{\epsilon}_{1:t-1})} \left[ \text{Cov}(B^\top \boldsymbol{z}_t \,|\, \boldsymbol{z}_{t-1}(\boldsymbol{\epsilon}_{1:t-1})) \right] \right\},$$

where the matrix $B = V \sqrt{\nabla^2 A(V^\top \boldsymbol{h}_t)}$ is the original prediction matrix $V$ rescaled by the square root of the Hessian of the log-normalizer, the inverse Fisher information matrix of the underlying RNN. This regularization term forces the hidden units to be robust to noise. Under weak unbiasedness, the proof holds under the assumption that the true data generating distribution is an RNN.

## Acknowledgements

We thank Francisco Ruiz for presenting our paper at ICML, 2018. We thank the Princeton Institute for Computational Science and Engineering (PICSciE), the Office of Information Technology's High Performance Computing Center and Visualization Laboratory at Princeton University for the computational resources. This work was supported by ONR N00014-15-1-2209, ONR 133691-5102004, NIH 5100481-5500001084, NSF CCF-1740833, the Alfred P. Sloan Foundation, the John Simon Guggenheim Foundation, Facebook, Amazon, and IBM.

## References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bayer, J. and Osendorfer, C. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Bishop, C. M. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

Chiu, C.-C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R. J., Rao, K., Gonina, K., et al. State-of-the-art speech recognition with sequence-to-sequence models. *arXiv preprint arXiv:1712.01769*, 2017.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, pp. 2980–2988, 2015.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

Elman, J. L. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Fraccaro, M., Sønderby, S. K., Paquet, U., and Winther, O. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pp. 2199–2207, 2016.

Gal, Y. and Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1019–1027, 2016.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

Goyal, A., Sordoni, A., Côté, M.-A., Ke, N., and Bengio, Y. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pp. 6716–6726, 2017.

Grave, E., Joulin, A., and Usunier, N. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Graves, A., Mohamed, A.-r., and Hinton, G. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649. IEEE, 2013.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Inan, H., Khosravi, K., and Socher, R. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Jim, K.-C., Giles, C. L., and Horne, B. G. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on Neural Networks*, 7(6):1424–1438, 1996.

Krause, B., Kahembwe, E., Murray, I., and Renals, S. Dynamic evaluation of neural sequence models. *arXiv preprint arXiv:1709.07432*, 2017.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Larochelle, H., Courville, A., et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Mikolov, T. and Zweig, G. Context dependent recurrent neural network language model. *SLT*, 12:234–239, 2012.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Interspeech*, volume 2, pp. 3, 2010.

Noh, H., You, T., Mun, J., and Han, B. Regularizing deep neural networks by noise: Its interpretation and optimization. In *Advances in Neural Information Processing Systems*, pp. 5113–5122, 2017.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, 28:1310–1318, 2013.

Pearlmutter, B. A. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

Poggio, T., Rifkin, R., Mukherjee, S., and Rakhlin, A. Bagging regularizes. Technical report, Massachusetts Institute of Technology, 2002.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Robbins, H. The empirical bayes approach to statistical decision problems. *The Annals of Mathematical Statistics*, 35(1):1–20, 1964.

Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

Robinson, A. and Fallside, F. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

Semeniuta, S., Severyn, A., and Barth, E. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

Wager, S., Wang, S., and Liang, P. S. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, pp. 351–359, 2013.

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066, 2013.

Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

Williams, R. J. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical report, Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: A high-rank RNN language model. *arXiv preprint arXiv:1711.03953*, 2017.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.