
On Fast Dropout and its Applicability to Recurrent Networks

**Justin Bayer, Christian Osendorfer, Daniela Korhammer,
Nutan Chen, Sebastian Urban and Patrick van der Smagt**

Lehrstuhl für Robotik und Echtzeitsysteme

Fakultät für Informatik

Technische Universität München

bayer.justin@googlemail.com, osendorf@in.tum.de, korhammd@in.tum.de,
ntchen86@gmail.com, surban@tum.de, smagt@brml.org

Abstract

Recurrent Neural Networks (RNNs) are rich models for the processing of sequential data. Recent work on advancing the state of the art has been focused on the optimization or modelling of RNNs, mostly motivated by addressing the problems of the vanishing and exploding gradients. The control of overfitting has seen considerably less attention. This paper contributes to that by analyzing fast dropout, a recent regularization method for generalized linear models and neural networks from a back-propagation inspired perspective. We show that fast dropout implements a quadratic form of an adaptive, per-parameter regularizer, which rewards large weights in the light of underfitting, penalizes them for overconfident predictions and vanishes at minima of an unregularized training loss. The derivatives of that regularizer are exclusively based on the training error signal. One consequence of this is the absence of a global weight attractor, which is particularly appealing for RNNs, since the dynamics are not biased towards a certain regime. We positively test the hypothesis that this improves the performance of RNNs on four musical data sets.

1 Introduction

Recurrent Neural Networks are among the most powerful models for sequential data. The capability of representing any measurable sequence to sequence mapping to arbitrary accuracy (Hammer, 2000) makes them universal approximators. Nevertheless they were given only little attention in the last two decades due to the problems of vanishing and exploding gradients (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2012). Error signals either blowing up or decaying exponentially for events many time steps apart rendered them largely impractical for the exact problems they were supposed to solve. This made successful training impossible on many tasks up until recently without resorting to special architectures or abandoning gradient-based optimization. Successful application on tasks with long-range dependencies has thus relied on one of those two paradigms. The former is to make use of long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). These approaches are among the best methods for the modelling of speech and handwriting (Graves et al., 2013, 2008; Graves, 2013). The latter is to rely on sensible initializations leading to echo-state networks (Jäger et al., 2003).

The publication of (Martens and Sutskever, 2011) can nowadays be considered a landmark, since it was shown that even standard RNNs can be trained with the right optimization method. While a sophisticated Hessian-free optimizer was employed initially, further research (Sutskever et al., 2013; Bengio et al., 2012) has shown that carefully designed first-order methods can find optima of similar quality.

After all, the problem of underfitting standard RNNs can be dealt with to the extent that RNNs are practical in many areas, e.g., language modelling (Sutskever et al., 2011; Mikolov et al., 2010). In contrast, the problem of overfitting in standard RNNs has (due to the lack of necessity) been tackled only by few. As noted in (Pascanu et al., 2012), using priors with a single optima on the parameters may have detrimental effects on the representation capability of RNNs: a global attractor is constructed in parameter space. In the case of a prior with a mode at zero (e.g. an L_2 regularizer) this biases the network towards solutions which lets information die out exponentially fast in time, making it impossible to memorize events for an indefinite amount of time.

Graves (2011) proposes to stochastically and adaptively distort the weights of LSTM-based RNNs, which is justified from the perspective of variational Bayes and the minimum description length principle. Overfitting is practically non-existent in the experiments conducted. It is untested whether this approach works well for standard RNNs—along the lines of the observations of Pachitariu and Sahani (2013) one might hypothesize that the injected noise disturbs the dynamics of RNNs too much and leads to divergence during training.

The deep neural network community has recently embraced a regularization method called dropout (Hinton et al., 2012). The gist is to randomly discard units from the network during training, leading to less interdependent feature detectors in the intermediary layers. Here, dropping out merely means to set the output of that unit to zero. An equivalent view is to set the complete outgoing weight vector to zero of which it is questionable whether a straight transfer of dropout to RNNs is possible. The resulting changes to the dynamics of an RNN during every forward pass are quite dramatic. This is the reason why Pachitariu and Sahani (2013) only use dropout on those parts of the RNN which are not dynamic., i.e. the connections feeding from the hidden into the output layer.

Our contribution is to show that using a recent smooth approximation to dropout (Wang and Manning, 2013) regularizes RNNs effectively. Since the approximation is deterministic, we may assert that all dynamic parts of the network operate in reasonable regimes. We show that fast dropout does not keep RNNs from reaching rich dynamics during training, which is not obvious due to the relation of classic dropout to L_2 regularization (Wager et al., 2013).

The structure of the paper is as follows. We will first review RNNs and fast dropout (FD) (Wang and Manning, 2013). A novel analysis of the derivatives of fast dropout leads to an interpretation where we can perform a decomposition into a loss based on the average output of a network’s units and a regularizer based on its variance. We will discuss why this is a form that is well suited to RNNs and consequently conduct experiments that confirm our hypothesis.

2 Methods

In this section we will first review RNNs and fast dropout. We will then introduce a novel interpretation of what the fast dropout loss constitutes in section 2.2.2 and show relationships to two other regularizers.

2.1 Recurrent Neural Networks

We will define RNNs in terms of two components. For one, we are ultimately interested in an output \mathbf{y} , which we can calculate given the parameters θ of a network and some input \mathbf{x} . Secondly, we want to learn the parameters, which is done by the design and optimization of a function of the parameters $\mathcal{L}(\theta)$, commonly dubbed loss, cost, or error function.

Calculating the Output of an RNN Given an input sequence $\mathbf{x} = (x_1, \dots, x_T), x_t \in \mathbb{R}^\kappa$ we produce an output $\mathbf{y} = (y_1, \dots, y_T), y_t \in \mathbb{R}^\omega$ which is done via an intermediary representation called the hidden state layer $\mathbf{h} = (h_1, \dots, h_T), h_t \in \mathbb{R}^\gamma$. κ, ω , and γ are the dimensionalities of the inputs, outputs, and hidden state at each time step. Each component of the layers is sometimes referred to as a unit or a “neuron”. Depending on the associated layer, these are then input, hidden, or output units. We will also denote the set of units which feed into some unit i as the incoming units of i . The units into which a unit i feeds are called the outgoing units of i . For a recurrent network

with a single hidden layer, this is done via iteration of the following equations from $t = 1$ to T :

$$\begin{aligned} h_t &= f_h(x_t W_{\text{in}} + h_{t-1} W_{\text{rec}} + b_h), \\ y_t &= f_y(h_t W_{\text{out}} + b_y), \end{aligned}$$

where $\{W_{\text{in}}, W_{\text{out}}, W_{\text{rec}}\}$ are weight matrices and $\{b_h, b_y\}$ bias vectors. These form the set of parameters θ together with initial hidden state h_0 . The dimensionalities of all weight matrices, bias vectors, and initial hidden states are determined by the dimensionalities of the input sequences as well as desired hidden layer and output layer sizes. The functions f_h and f_y are so-called transfer functions and mostly coordinate-wise applied nonlinearities. We will call the activations of units *pre-synaptic* before the application of f and *post-synaptic* afterwards. Typical choices include the *logistic sigmoid* $f(\xi) = \frac{1}{1+\exp(-\xi)}$, *tangent hyperbolicus* and, more recently, the *rectified linear* $f(\xi) = \max(\xi, 0)$ (Zeiler et al., 2013). If we set the recurrent weight matrix W_{rec} to zero, we recover a standard neural network Bishop (1995) applied independently to each time step.

Loss Function and Adaption of Parameters We will restrict ourselves to RNNs for the supervised case, where we are given a data set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{z}_i)\}_{i=1}^N$ consisting of N pairs with $\mathbf{x}_i \in \mathbb{R}^{T \times \kappa}$ and $\mathbf{z}_i \in \mathbb{R}^{T \times \omega}$. Here T refers to the sequence length, which we assume to be constant over the data set. We are interested to adapt the parameters of the network θ in a way to let each of its outputs $\mathbf{y}_i \in \mathbb{R}^{T \times O}$ be close to \mathbf{z}_i . Closeness is typically formulated as a loss function, e.g. the mean squared error $\mathcal{L}_{\text{mse}}(\theta) = \sum_i \|\mathbf{z}^{(i)} - \mathbf{y}^{(i)}\|_2^2$ or the binary cross entropy $\mathcal{L}_{\text{bce}}(\theta) = \sum_i z^{(i)} \log y^{(i)} + (1 - z^{(i)}) \log(1 - y^{(i)})$. If a loss \mathcal{L} is locally differentiable, finding good parameters can be performed by gradient-based optimization, such as nonlinear conjugate gradients or stochastic gradient descent. The gradients can be calculated efficiently via back-propagation through time (BPTT) (Rumelhart et al., 1986).

2.2 Fast Dropout

In fast dropout (Wang and Manning, 2013), each unit in the network is assumed to be a random variable. To assure tractability, only the first and second moments of those random variables are kept, which suffices for a very good approximation. Since the pre-synaptic activation of each unit is a weighted sum of its incoming units (of which each is dropped out with a certain probability) we can safely assume Gaussianity for those inputs due to the central limit theorem. As we will see, this is sufficient to find efficient ways to propagate the mean and variance through the non-linearity f .

2.2.1 Forward propagation

We will now inspect the forward propagation for a layer into a single unit, that is

$$\begin{aligned} a &= (\mathbf{d} \circ \mathbf{x})^T \mathbf{w} \\ y &= f(a), \end{aligned}$$

where \circ denotes the element-wise product and f is a non-linear transfer function as before. Let the input layer \mathbf{x} to the unit be Gaussian distributed with diagonal covariance by assumption: $\mathbf{x} \sim \mathcal{N}(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}^2)$. Furthermore, we have Bernoulli distributed variables indicating whether an incoming unit is not being dropped out organized in a vector \mathbf{d} with $d_i \sim \mathcal{B}(p)$, p being the complementary drop out rate. The weight vector \mathbf{w} is assumed to be constant.

A neural network will in practice consist of many such nodes, with some of them, the output units, directly contributing to the loss function \mathcal{L} . Others, the input units, will not stem from calculation but come from the data set. Each component of \mathbf{x} represents an incoming unit, which might be an external input to the network or a hidden unit. In general, y will have a complex distribution depending highly on the nature of f . Given that the input to a function f is Gaussian distributed, we obtain the mean and variance of the output as follows:

$$\begin{aligned} \mathbb{E}[y] = f_{\mu}(a) &= \int f(x) \mathcal{N}(x | \mathbb{E}[a], \mathbb{V}[a]) dx, \\ \mathbb{V}[y] = f_{\sigma}(a) &= \int (f(x) - f_{\mu}(a))^2 \mathcal{N}(x | \mathbb{E}[a], \mathbb{V}[a]) dx. \end{aligned}$$

Forward propagation through the non-linearity f for calculation of the post-synaptic activation can be approximated very well in the case of the *logistic sigmoid* and the *tangent hyperbolicus* and done exactly in case of the *rectifier* (for details, see Wang and Manning (2013)). While the rectifier has been previously reported to be a useful ingredient in RNNs (Bengio et al., 2012) we found that it leads to unstable learning behaviour in preliminary experiments and thus neglected it in this study, solely focusing on the tangent hyperbolicus. Other popular transfer functions, such as the softmax, need to be approximated either via sampling or an unscented transform (Julier and Uhlmann, 1997).

To obtain a Gaussian approximation for $a = (\mathbf{d} \circ \mathbf{x})^T \mathbf{w}$, we will use $\hat{a} \sim \mathcal{N}(\mathbb{E}[a], \mathbb{V}[a])$. The mean and variance of a can be obtained as follows. Since \mathbf{d} and \mathbf{x} are independent it follows that

$$\mathbb{E}[a] = \mathbb{E}[(\mathbf{x} \circ \mathbf{d})^T \mathbf{w}] = (\mathbb{E}[\mathbf{x}] \circ \mathbb{E}[\mathbf{d}])^T \mathbf{w}. \quad (1)$$

For independent random variables A and B , $\mathbb{V}[AB] = \mathbb{V}[A]\mathbb{E}[B]^2 + \mathbb{E}[A]^2\mathbb{V}[B] + \mathbb{V}[A]\mathbb{V}[B]$. If we assume the components of \mathbf{x} to be independent, we can write¹

$$\mathbb{V}[a] = (p(1-p)\mu_{\mathbf{x}}^2 + p\sigma_{\mathbf{x}}^2)^T \mathbf{w}^2. \quad (2)$$

Furthermore the independency assumption is necessary such that the Lyapunov condition is satisfied (Lehmann, 1999) for the the central limit theorem to hold, ensuring that a is approximately Gaussian.

Propagating the mean and the variance through f via f_μ and f_σ suffices for determining the pre-synaptic moments of the outgoing units. At the output \mathbf{y} of the whole model, we will simplify matters and ignore the variance. Some loss functions take the variance into account (e.g., a Gaussian log-likelihood as done in (Bayer et al., 2013)). Sampling can be a viable alternative as well.

Fast Dropout for RNNs The extension of fast dropout to recurrent networks is straightforward from a technical perspective. First, we note that we can concatenate the input vector at time step t , x_t and the hidden state at the previous layer h_{t-1} into a single vector: $c_t = [h_{t-1}, x_t]$. We obtain a corresponding weight matrix by concatenation of the input to hidden and recurrent weight matrices W_{in} and W_{rec} : $W_c = [W_{\text{rec}}, W_{\text{in}}]$. We can thus reduce the computation to the step from above.

2.2.2 Beyond the Backward Pass: A Regularization Term

Given the forward pass, we used automatic differentiation with Theano (Bergstra et al., 2010) to calculate the gradients. Nevertheless, we will contribute a close inspection of the derivatives. This will prove useful since it makes it possible to interpret fast dropout as an additional regularization term *independent of the exact choice of loss function*.

Consider a loss $\mathcal{L}(\mathcal{D}; \theta)$ which is a function of the data \mathcal{D} and parameters θ ². In machine learning, we wish this loss to be minimal under unseen data \mathcal{D} although we only have access to a training set $\mathcal{D}_{\text{train}}$. A typical approach is to optimize another loss $\mathcal{J}(\mathcal{D}_{\text{train}}; \theta)$ as a proxy in the hope that a good minimum of it will correspond to a good minimum of \mathcal{L} for unseen data. Learning is often done by the optimization of $\mathcal{J} = \mathcal{L} + \mathcal{R}$, where \mathcal{R} is called a regularizer. A common example of a regularizer is to place a prior on the parameters, in which case it is a function of θ and corresponds to the log-likelihood of the parameters. For weight decay, this is a spherical Gaussian with inverse scale λ , i.e. $\mathcal{R}_{\text{wd}}(\theta) = \lambda \|\theta\|_2^2$. Regularizers can be more sophisticated, e.g. Rifai et al. (2011) determine directions in input space to which a model’s outputs should be invariant. More recently, dropout (i.e. non-fast dropout) for generalized linear models has been interpreted as a semi-supervised regularization term encouraging confident predictions by Wager et al. (2013).

While it seems difficult to bring the objective function $\mathcal{J}_{\text{fd}}(\mathcal{D}; \theta)$ of fast dropout into the form of $\mathcal{L} + \mathcal{R}$, it is possible with the derivatives of each node. For this, we perform back-propagation like calculations.

Let $a = (\mathbf{d} \circ \mathbf{x})^T \mathbf{w}$ and $y = f(a)$ be the pre- and post-synaptic activations of a component of a layer in the network. First note that $\partial \mathcal{J} / \partial w_i = \partial \mathcal{J} / \partial a \cdot \partial a / \partial w_i$ according to the chain rule. Since a is a random variable, it will be described in one of two forms. In the case of a Gaussian approximation, we will summarize it in terms of its mean and variance; this approach is used if propagation through f is possible in closed form. In the case of sampling, we will have a single instantiation \hat{a} of the random variable, which we can propagate through f . An analysis of both cases is as follows.

¹In contrast to Wang and Manning (2013), we do not drop the variance not related to dropout. In their case, $\mathbb{V}[a] = p(1-p)\mu_{\mathbf{x}}^2 \mathbf{w}^2$ was used instead of Equation (2).

²We will frequently omit the explicit dependency on \mathcal{D} and θ where clear from context.

Gaussian approximation We find the derivative of \mathcal{J} with respect to one of its incoming weights w_i to be

$$\frac{\partial \mathcal{J}}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \mathbb{E}[a]} \frac{\partial \mathbb{E}[a]}{\partial w_i} + \frac{\partial \mathcal{J}}{\partial \mathbb{V}[a]} \frac{\partial \mathbb{V}[a]}{\partial w_i}.$$

We know that $\mathbb{E}[a] = (\mathbf{x} \circ \mathbf{d})^T \mathbf{w}$ and thus $\partial \mathbb{E}[a] / \partial w_i = x_i d_i$. This can be recognized as the standard back-propagation term if we consider the dropout variable d_i as fixed. We will thus define

$$\frac{\partial \mathcal{L}^a}{\partial w_i} := \frac{\partial \mathcal{J}}{\partial \mathbb{E}[a]} \frac{\partial \mathbb{E}[a]}{\partial w_i}, \quad (3)$$

and subsequently refer to it as the local derivative of the training loss. The second term can be analysed similarly. We apply the chain-rule once more which yields

$$\frac{\partial \mathcal{J}}{\partial \mathbb{V}[a]} \frac{\partial \mathbb{V}[a]}{\partial w_i} = \underbrace{\frac{\partial \mathcal{J}}{\partial \mathbb{V}[a]} \frac{\partial \mathbb{V}[a]}{\partial w_i^2}}_{:= \delta^a} \frac{\partial w_i^2}{\partial w_i}.$$

for which any further simplification of δ^a depends on the exact form of \mathcal{J} . The remaining two factors can be written down explicitly, i.e.

$$\begin{aligned} \frac{\partial \mathbb{V}[a]}{\partial w_i^2} &= p(1-p)\mathbb{E}[x_i]^2 + p\mathbb{V}[x_i], \\ \frac{\partial w_i^2}{\partial w_i} &= 2w_i. \end{aligned}$$

Setting

$$\begin{aligned} \eta_i^a &:= |\delta^a| \frac{\partial \mathbb{V}[a]}{\partial w_i^2} \\ &= |\delta^a| p [(1-p)\mathbb{E}[x_i]^2 + \mathbb{V}[x_i]] \\ &> 0 \end{aligned}$$

we conclude that

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial \mathbb{V}[a]} \frac{\partial \mathbb{V}[a]}{\partial w_i} &= 2 \operatorname{sgn}(\delta^a) \eta_i^a w_i \\ &=: \frac{\partial \mathcal{R}_{\text{approx}}^a}{\partial w_i}. \end{aligned}$$

In alignment with Equation (3) this lets us arrive at

$$\frac{\partial \mathcal{J}}{\partial w_i} = \frac{\partial \mathcal{L}^a}{\partial w_i} + \frac{\partial \mathcal{R}_{\text{approx}}^a}{\partial w_i},$$

and offers an interpretation of fast dropout as an additive regularization term. An important and limiting aspect of this decomposition is that it only holds locally at a .

We note that depending on the sign of the error signal δ^a , fast dropout can take on three different behaviours:

- $\delta^a = 0$ The error signal is zero and thus the variance of the unit considered to be optimal for the loss. The fast dropout term vanishes; this is especially true at optima of the overall loss.
- $\delta^a < 0$ The unit should increase its variance. The exact interpretation of this depends on the loss, but in many cases this is related to the expectation of the unit being quite erroneous and leads to an increase of scatter of the output. The fast dropout term encourages a quadratic growth of the weights.
- $\delta^a > 0$ The unit should decrease its variance. As before, this depends on the exact loss function but will mostly be related to the expectation of the unit being quite right which makes a reduction of scatter desirable. The fast dropout term encourages a quadratic shrinkage of the weights.

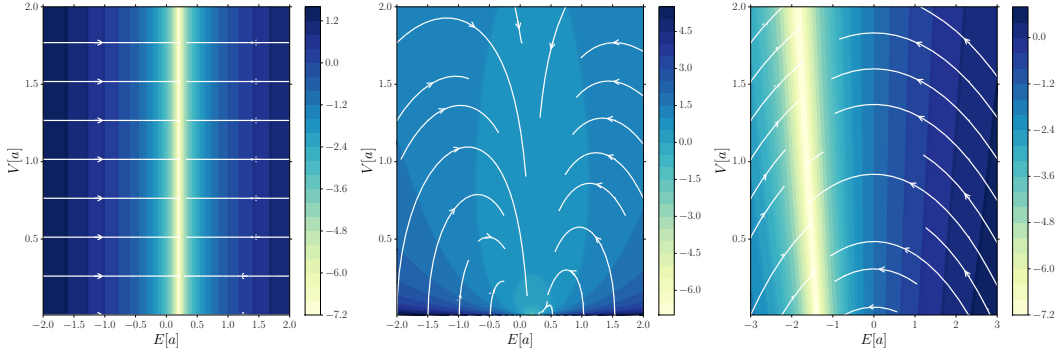


Figure 1: Visualizations of the behaviour of δ^a for a single unit. The axes correspond to the pre-synaptic mean $E[a]$ and variance $V[a]$ feeding into a unit $y = f(a)$. A loss measuring the divergence from the target value 0.2 is then applied and indices the color on a logarithmic scale. The gradients of the loss are shown as a vector field plot. Squared error is shown on the left, Gaussian log-likelihood in the middle and Bernoulli log-likelihood on the right. For the first two plots, the optimum is in the middle, for the last it is a little to the left.

This behaviour can be illustrated for output units by numerically inspecting the values and gradients of the pre-synaptic moments given a loss. For that we consider a single unit $y = f(a)$ and a loss $d(y, z)$ measuring the divergence of its output to a target value z . The pre-synaptic variance $V[a]$ can enter the loss not at all or in one of two ways, respected by either the loss (see (Bayer et al., 2013)) or the transfer function. Three examples for this are

1. Squared loss on the mean, i.e. $d(y, z) = (E[y] - t)^2$ with $y = a$,
2. Gaussian log-likelihood on the moments, i.e. $d(y, z) \propto \frac{(E[y] - t)^2}{2V[y]} + \log \sqrt{2\pi V[y]}$ with $y = a$,
3. Negative Bernoulli cross entropy, i.e. $d(y, z) = z \log E[y] + (1 - z) \log(1 - E[y])$ with $y = \frac{1}{1 + \exp(-a)}$.

We visualize the pre-synaptic mean and variance, their gradients and their respective loss values in Figure 1. For the two latter cases, erroneous units first increase the variance, then move towards the correct mean and subsequently reduce the variance.

Sampling An already mentioned alternative to the forward propagation of $E[a]$ and $V[a]$ through f is to incarnate \hat{a} via sampling and calculate $f(\hat{a})$. Let $s \sim \mathcal{N}(0, 1)$ and $\hat{a} = E[a] + s\sqrt{V[a]}$. We can then use \hat{a} explicitly and it follows that

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \hat{a}}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} \left[\frac{\partial \hat{a}}{\partial E[a]} \frac{\partial E[a]}{\partial w_i} + \frac{\partial \hat{a}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i} \right].$$

Again, we recognize $\frac{\partial \mathcal{J}}{\partial \hat{a}} \cdot \frac{\partial \hat{a}}{\partial E[a]} \cdot \frac{\partial E[a]}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} x_i d_i$ as the standard back-propagation formula with dropout variables.

The variance term can be written as

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \mathcal{J}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \hat{a}}{\partial \sqrt{V[a]}} \frac{\partial \sqrt{V[a]}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i^2} \frac{\partial w_i^2}{\partial w_i}$$

which, making use of results from earlier in the section is equivalent to

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \mathcal{J}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} \sqrt{p(1-p)E[x_i]^2 + pV[x_i]} s. \quad (4)$$

The value of this is a zero-centred Gaussian random variable, since s is Gaussian. The scale is independent of the current weight value and only determined by the post-synaptic moments of the incoming unit, the dropout rate and the error signal. We conclude, that also in this case, we can write

$$\frac{\partial \mathcal{J}}{\partial w_i} = \frac{\partial \mathcal{L}^a}{\partial w_i} + \frac{\partial \mathcal{R}_{\text{draw}}^a}{\partial w_i},$$

where $\partial\mathcal{R}_{\text{draw}}^a/\partial w_i$ is defined as in Equation (4) and essentially an adaptive noise term.

We want to stress the fact that in the approximation as well as in the sampling case the regularization term vanishes at any optima of the training loss. A consequence of this is that no global attractor is formed, which makes the method theoretically useful for RNNs. One might argue that fast dropout should not have a regularizing effect all. Yet, regularization is not only influencing the final solution but also the optimization, leading to different optima.

Relationship to Weight Decay As already mentioned, imposing a Gaussian distribution centred at the origin with precision λ as a prior on the weights leads to a method called weight decay. It is not only probabilistically sound but also works well empirically, see e.g. (Bishop et al., 2006). Recalling that the derivative of weight decay is of the form $\partial\mathcal{R}_{\text{wd}}/\partial w_i = 2\lambda w_i$ we can reinterpret $\mathcal{R}_{\text{approx}}$ as a weight decay term where the coefficient λ is weight-wise, dependent on the current activations and possibly negative. Weight decay will always be slightly wrong on the training set, since the derivative of the weight decay term has to match the one of the unregularized loss. In order for $\partial\mathcal{L}/\partial\theta + \partial\mathcal{R}_{\text{wd}}/\partial\theta$ to be minimal, \mathcal{L} cannot be minimal unless so is \mathcal{R} .

Relationship to Adaptive Weight Noise In this method (Graves, 2011) not the units but the weights are stochastic, which is in practice implemented by performing Monte Carlo sampling. We can use a similar technique to FD to find a closed form approximation. In this case, a layer is $y = f(\mathbf{x}^T \mathbf{w})$, where we have no dropout variables and the weights are Gaussian distributed with $\mathbf{w} \sim \mathcal{N}(\mu_{\mathbf{w}}, \sigma_{\mathbf{w}}^2)$, with covariance diagonal and organized into a vector. We assume Gaussian density for $a = \mathbf{x}^T \mathbf{w}$. Using similar algebra as above, we find that

$$\begin{aligned} \text{E}[a] &= \mu_{\mathbf{x}}^T \mu_{\mathbf{w}}, & (5) \\ \text{V}[a] &= \sigma_{\mathbf{x}}^{2T} \mu_{\mathbf{w}}^2 + \sigma_{\mathbf{x}}^{2T} \sigma_{\mathbf{w}}^2 + \mu_{\mathbf{x}}^{2T} \sigma_{\mathbf{w}}^2. & (6) \end{aligned}$$

It is desirable to determine whether fast dropout and “fast adaptive weight noise” are special cases of each other. Showing that the approaches are different can be done by equating equations (1) and (5) and solving for $\mu_{\mathbf{w}}$. This shows that rescaling by $1 - p$ suffices in the case of the expectation. It is however not as simple as that for the variance, i.e. for equations (2) and (6), where the solution depends on $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}^2$ and thus is not independent of the input to the network. Yet, both methods share the property that no global attractor is present in the loss: the “prior” is part of the optimization and not fixed.

2.3 Bag of Tricks

Throughout the experiments we will resort to several “tricks” that have been introduced recently for more stable and efficient optimization of neural networks and RNNs especially. First, we make use of *rmsprop* (Tieleman and Hinton, 2012), an optimizer which divides the gradient by an exponential moving average of its squares. This approach is similar to Adagrad (Duchi et al., 2011), which uses a window based average. We found that enhancing *rmsprop* with Nesterov’s accelerated gradient (Sutskever, 2013) greatly reduces the training time in preliminary experiments.

To initialize the RNNs to stable dynamics we followed the initialization protocol of (Sutskever et al., 2013) of setting the spectral radius ρ to a specific value and the maximum amount of incoming connections of a unit to ν ; we did not find it necessary to centre the inputs and outputs. The effect of not only using the recurrent weight matrix for propagating the states through time but also its element-wise square for advancing the variances can be quantified. The stability of a network is coupled to the spectral radius of the recurrent weight matrix $\rho(W_{\text{rec}})$; thus, the stability of forward propagating the variance is related to the spectral radius of its element-wise square $\rho(W_{\text{rec}}^2)$. Since $\rho(AB) \leq \rho(A)\rho(B)$ for non-negative matrices and non-singular matrices A and B (Horn and Johnson, 2012), setting W_{rec} to full rank and its spectral radius to $\rho(W_{\text{rec}})$ assures $\rho(W_{\text{rec}}^2) = \rho(|W_{\text{rec}}|^2) \leq \rho(W_{\text{rec}})^2$, where $|\cdot|$ denotes taking the absolute value element-wise. We also use the gradient clipping method introduced in (Pascanu et al., 2012), with a fixed threshold of 225.

Since the hidden-to-hidden connections and the hidden-to-output connections in an RNN can make use of hidden units in quite distinct ways, we found it beneficial to separate the dropout rates. Specifically, a hidden unit may have a different probability to be dropped out when feeding into

Table 1: Results on the midi data sets. All numbers are average negative log-likelihoods on the test set, where “FD” represents our work; “plain” and “RNN-NADE” results are from (Bengio et al., 2012) while “Deep RNN” shows the best results from (Pascanu et al., 2013). Note that “RNN-NADE” and “Deep RNN” employ various extensions of the model structure of this work, i.e. structured outputs and various forms of depths. Our results are the best for the shallow RNN model considered in this work.

Data set	FD	plain	RNN-NADE	Deep RNN
Piano-midi.de	7.39	7.58	7.05	–
Nottingham	3.09	3.43	2.31	2.95
MuseData	6.75	6.99	5.60	6.59
JSBChorales	8.01	8.58	5.19	7.92

the hidden layer at the next time step than when feeding into the output layer. Taking this one step further, we also consider networks in which we completely neglect fast dropout for the hidden-to-output connections; an ordinary forward pass is used instead. Note that this is not the same as setting the dropout rate to zero, since the variance of the incoming units is completely neglected. Whether this is done is treated as another hyperparameter for the experiment.

3 Experiments and Results

3.1 Musical Data

All experiments were done by performing a random search (Bergstra and Bengio, 2012) over the hyper parameters (see Table 2 in the Appendix for an overview), where 32 runs were performed for each data set. We report the test loss of the model with the lowest validation error over all training runs, using the same split as in (Bengio et al., 2012). To improve speed, we organize sequences into minibatches by first splitting all sequences of the training and validation set into chunks of length of 100. Zeros are prepended to those sequences which have less than 100 time steps. The test error is reported on the unsplit sequences.

Training RNNs to generatively model polyphonic music is a valuable benchmark for RNNs due to its high dimensionality and the presence of long as well as short term dependencies. This data set has been evaluated previously by Bengio et al. (2012) where the model achieving the best results, RNN-NADE (Boulanger-Lewandowski et al., 2013), makes specific assumptions about the data (i.e. binary observables). RNNs do not attach any assumptions to the inputs.

3.1.1 Setup

The data consists of four distinct data sets, namely *Piano-midi.de* (classical piano music), *Nottingham* (folk music), *MuseData* (orchestral) and *JSBChorales* (chorales by Johann Sebastian Bach). Each has a dimensionality of 88 per time step organized into different *piano rolls* which are sequences of binary vectors; each component of these vectors indicates whether a note is occurring at the given time step. We use the RNN’s output to model the sufficient statistics of a Bernoulli random variable, i.e.

$$p(x_{t,i} | \mathbf{x}_{1:t-1}) = y_{t,i},$$

which describes the probability that note i is present at time step t . The output non-linearity f_y of the network is a sigmoid which projects the points to the interval $(0, 1)$. We perform learning by the minimization of the average negative log-likelihood (NLL); in this case, this is the average binary cross-entropy

$$\mathcal{L}(\theta) = \frac{1}{T-1} \frac{1}{N} \sum_{i,t,k} x_{t,i}^{(k)} \log y_{t-1,i}^{(k)} + (1 - x_{t,i}^{(k)}) \log(1 - y_{t-1,i}^{(k)}),$$

where k indices the training sample, i the component of the target and t the time step.

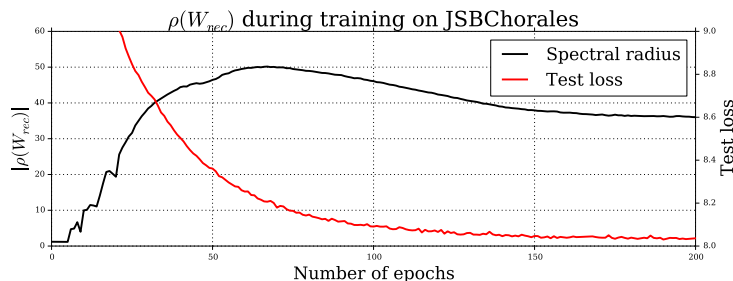


Figure 2: Spectral radius over the training process. It increases first and then slowly decreases until a certain value is reached. We did not observe this behaviour when training plain RNNs.

3.1.2 Results

Although a common metric for evaluating the performance of such benchmarks is that of accuracy (Bay et al., 2009) we restrict ourselves to that of the NLL—the measure of accuracy is not what is optimized and to which the NLL is merely a proxy. We present the results of FD-RNNs compared with the various other methods in Table 1. Our method is only surpassed by methods which either incorporate more specific assumptions of the data or employ various forms of depth (Boulanger-Lewandowski et al., 2013; Pascanu et al., 2013). We want to stress that we performed only 32 runs for each data set once more. This shows the relative ease to obtain good results despite of the huge space of potential hyper parameters.

One additional observation is the range of the Eigenvalues of the recurrent weight matrix W_{rec} during training. We performed an additional experiment on *JSBChorales* where we inspected the Eigenvalues and the test loss. We found that the spectral radius first increases sharply to a rather high value and then decreases slowly to settle to a specific value. We tried to replicate this behaviour in plain RNNs, but found that RNNs never exceeded a certain spectral radius at which they stuck. This stands in line with the observation from Section 2.2.2 that weights are encouraged to grow when the error is high and shrink during convergence to the optimum. See Figure 2 for a plot of the spectral radius $\rho(W_{\text{rec}})$ over the training process stages.

4 Conclusion

We have contributed to the field of neural networks in two ways. First, we have analysed a fast approximation of the dropout regularization method by bringing its derivative into the same form as that of a loss regularized with an additive term. We have used this form to gain further insights upon the behaviour of fast dropout for neural networks in general and shown that this objective function does not bias the solutions to those which perform suboptimal on the unregularized loss. Second, we have hypothesized that this is beneficial especially for RNNs. We confirmed this hypothesis by conducting quantitative experiments on an already established benchmark used in the context of learning recurrent networks.

References

- Bay, M., Ehmann, A. F., and Downie, J. S. (2009). Evaluation of multiple-f0 estimation and tracking systems. In *ISMIR*, pages 315–320.
- Bayer, J., Osendorfer, C., Urban, S., et al. (2013). Training neural networks with implicit variance. In *Proceedings of the 20th International Conference on Neural Information Processing, ICONIP-2013*.
- Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2012). Advances in optimizing recurrent networks. *arXiv preprint arXiv:1212.0901*.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

- Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305.
- Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.
- Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.
- Bishop, C. M. et al. (2006). *Pattern recognition and machine learning*, volume 1. springer New York.
- Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2013). High-dimensional sequence transduction. In *ICASSP*.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 999999:2121–2159.
- Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356.
- Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.
- Graves, A., Fernández, S., Liwicki, M., Bunke, H., and Schmidhuber, J. (2008). Unconstrained online handwriting recognition with recurrent neural networks. *Advances in Neural Information Processing Systems*, 20:1–8.
- Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *arXiv preprint arXiv:1303.5778*.
- Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1):107–123.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.
- Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut für Informatik, Technische Universität, München*.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Horn, R. A. and Johnson, C. R. (2012). *Matrix analysis*. Cambridge university press.
- Jäger, H. et al. (2003). Adaptive nonlinear system identification with echo state networks. *networks*, 8:9.
- Julier, S. J. and Uhlmann, J. K. (1997). New extension of the kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics.
- Lehmann, E. L. (1999). *Elements of large-sample theory*. Springer Verlag.
- Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. *Proc. 28th Int. Conf. on Machine Learning*.
- Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. *Proceedings of Interspeech*.
- Pachitariu, M. and Sahani, M. (2013). Regularization and nonlinearities for neural language models: when are they needed? *arXiv preprint arXiv:1301.5650*.
- Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.
- Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. Technical report, Technical Report.
- Rifai, S., Dauphin, Y. N., Vincent, P., Bengio, Y., and Muller, X. (2011). The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pages 2294–2302.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Table 2: Hyper parameter ranges and parameter distributions for the musical data sets.

Hyper parameter	Choices
#hidden layers	1
#hidden units	200, 400, 600
Transfer function	tanh
p (“dropout input”)	0.0, 0.1, 0.2
p (“dropout hidden to hidden”)	0.0, 0.1, 0.2, 0.3, 0.4, 0.5
p (“dropout hidden to output”)	0.0, 0.2, 0.5
Use fast dropout for final layer	yes, no
Step rate	0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00001
Momentum	0.0, 0.9, 0.95, 0.99, 0.995
Decay	0.8, 0.9
W	$\mathcal{N}(0, \sigma^2), \sigma^2 \in \{0.1, 0.01, 0.001, 0.0001\}$
W_{in}	$\mathcal{N}(0, \sigma^2), \sigma^2 \in \{0.1, 0.01, 0.001, 0.0001\}$
b_h	0
b_y	-0.8
$\rho(W_{rec})$	1.0, 1.05, 1.1, 1.2
ν	15, 25, 35, 50 or no

- Sutskever, I. (2013). *Training Recurrent Neural Networks*. PhD thesis, University of Toronto.
- Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning.
- Sutskever, I., Martens, J., and Hinton, G. (2011). Generating text with recurrent neural networks. *Proceedings of the 2011 International Conference on Machine Learning (ICML-2011)*.
- Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.
- Wager, S., Wang, S., and Liang, P. (2013). Dropout training as adaptive regularization. *arXiv preprint arXiv:1307.1493*.
- Wang, S. and Manning, C. (2013). Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126.
- Zeiler, M., Ranzato, M., Monga, R., Mao, M., Yang, K., Le, Q., Nguyen, P., Senior, A., Vanhoucke, V., Dean, J., et al. (2013). On rectified linear units for speech processing. ICASSP.

5 Appendix

5.1 Hyper Parameters for Musical Data Experiments

We show the hyper parameter ranges for the musical data in Table 2. The ones from which the numbers in Table 1 resulted are given in Table 3.

Table 3: Hyper parameters used for the musical data experiments.

Hyper parameter	Piano-midi.de	Nottingham	MuseData	JSBChorales
#hidden units	600	400	600	400
p (“dropout input”)	0.1	0.1	0.2	0.1
p (“dropout hidden to hidden”)	0.3	0.4	0.3	0.2
p (“dropout hidden to output”)	–	0.0	0.0	0.5
Use fast dropout for final layer	no	yes	yes	yes
Step rate	0.005	0.001	0.0005	0.001
Momentum	0.995	0.99	0.995	0.99
Decay	0.8	0.9	0.9	0.8
σ^2 for $W_{\text{rec}}, W_{\text{out}}$	0.1	0.001	0.1	0.0001
σ^2 for W_{in}	0.0001	0.1	0.0001	0.01
$\rho(W_{\text{rec}})$	1.2	1.2	1.2	1.2
ν	25	no	no	15