Noisy Activation Functions

Caglar Gulcehre[†♯] GULCEHRC@IRO.UMONTREAL.CA
Marcin Moczulski[†◇] MARCIN.MOCZULSKI@STCATZ.OX.AC.UK
Misha Denil[†◇] MISHA.DENIL@GMAIL.COM
Yoshua Bengio[†♯] BENGIOY@IRO.UMONTREAL.CA
[†♯]University of Montreal
[†◇]University of Oxford

Abstract

Common activation functions used in neural networks can yield to training difficulties due to the saturation behavior of the activation function, which may hide dependencies that are not visible to vanilla-SGD (using first order gradients only). Gating mechanisms that use softly saturating activation functions to emulate the discrete switching of digital logic circuits are good examples of this. We propose to exploit the injection of appropriate noise so that the gradients may flow easily, even if the noiseless application of the activation function would yield zero gradient. Large noise will dominate the noise-free gradient and allow stochastic gradient descent to explore more. By adding noise only to the problematic parts of the activation function, we allow the optimization procedure to explore the boundary between the degenerate (saturating) and the well-behaved parts of the activation function. We also establish connections to simulated annealing, when the amount of noise is annealed down, making it easier to optimize hard objective functions. We find experimentally that replacing such saturating activation functions by noisy variants helps training in many contexts, yielding state-of-the-art or competitive results on different datasets and task, especially when training seems to be the most difficult, e.g., when curriculum learning is necessary to obtain good results.

1. Introduction

The introduction of the piecewise-linear activation functions such as ReLU and Maxout (Goodfellow et al., 2013) units had a profound effect on deep learning, and was a major catalyst in allowing the training of much deeper networks. It is thanks to ReLU that for the first time it was shown (Glorot et al., 2011) that deep purely supervised networks can be trained, whereas using $\tanh$ nonlinearity only allowed to train shallow networks. A plausible hypothesis about the recent surge of interest on these piecewise-linear activation functions (Glorot et al., 2011), is due to the fact that they are easier to optimize with SGD and backpropagation than smooth activation functions, such as $\mathrm{sigmoid}$ and $\tanh$. The recent successes of piecewise linear functions is particularly evident in computer vision, where the ReLU has become the default choice in convolutional networks.

We propose a new technique to train neural networks with activation functions which strongly saturate when their input is large. This is mainly achieved by injecting noise to the activation function in its saturated regime and learning the level of noise. Using this approach, we have found that it was possible to train neural networks with much wider family of activation functions than previously. Adding noise to the activation function has been considered for ReLU units and was explored in (Bengio et al., 2013; Nair & Hinton, 2010) for feed-forward networks and Boltzmann machines to encourage units to explore more and make the optimization easier.

More recently there has been a resurgence of interest in more elaborate "gated" architectures such as LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Cho et al., 2014), but also encompassing neural attention mechanisms that have been used in the NTM (Graves et al., 2014), Memory Networks (Weston et al., 2014), automatic image captioning (Xu et al., 2015) and wide areas of applications (LeCun et al., 2015). A common thread running through these works is the use of soft-saturating non-linearities, such as the sigmoid or softmax, to emulate the hard decisions of digital logic circuits. In spite of its success, there are two key problems with this approach.

1. Since the non-linearities still saturate there are problems with vanishing gradient information flowing through the gates; and

2. since the non-linearities only *softly* saturate they do not allow one to take hard decisions.

Although gates often operate in the soft-saturated regime (Karpathy et al., 2015; Bahdanau et al., 2014; Hermann et al., 2015) the architecture prevents them from being fully open or closed. We follow a novel approach to address both of these problems. Our method addresses the second problem through the use of hard-saturating nonlinearities, which allow gates to make perfectly on or off decisions when they saturate. Since the gates are able to be completely open or closed, no information is lost through the leakiness of the soft-gating architecture.

By introducing hard-saturating nonlinearities, we have exacerbated the problem of gradient flow, since gradients in the saturated regime are now precisely zero instead of being negligible. However, by introducing noise into the activation function which can grow based on the magnitude of saturation, we encourage random exploration.

At test time the noise in the activation functions can be removed or replaced with the expectation, and as our experiments show, the resulting deterministic networks outperform their soft-saturating counterparts on a wide variety of tasks, and allow to reach state-of-the-art performance by simple drop-in replacement of the nonlinearities in existing training code.

The technique that we propose, addresses the difficulty of optimization and having hard-activations at the test time for gating units and we propose a way of performing simulated annealing for neural networks.

2. Saturating Activation Functions

Definition 2.1. (Activation Function). An activation function is a function $\mathrm{h} : \mathcal{R} \to \mathcal{R}$ that is differentiable almost everywhere.

Definition 2.2. (Saturation). An activation function $\mathrm{h}(x)$ with derivative $\mathrm{h}'(x)$ is said to right (resp. left) saturate if its limit as $x \to \infty$ (resp. $x \to -\infty$) is zero. An activation function is said to saturate (without qualification) if it both left and right saturates.

Most common activation functions used in recurrent networks (for example, $\tanh$ and $\mathrm{sigmoid}$) are saturating. In particular they are soft saturating, meaning that they achieve saturation only in the limit.

Definition 2.3. Hard and Soft Saturation. Let c be a constant such that $x > c$ implies $\mathrm{h}'(x) = 0$ and left hard saturates when $x < c$ implies $\mathrm{h}'(x) = 0$, $\forall x$. We say that $\mathrm{h}(\cdot)$ hard saturates (without qualification) if it both left and right hard saturates. An activation function that saturates but achieves zero gradient only in the limit is said to soft saturate.

We can construct hard saturating versions of soft saturating activation functions by taking a first-order Taylor expansion about zero and clipping the results to an appropriate range.

For example, expanding $\tanh$ and $\mathrm{sigmoid}$ around 0, with $x \approx 0$, we obtain linearized functions u^t and u^s of $\tanh$ and $\mathrm{sigmoid}$ respectively:

$$\mathrm{sigmoid}(x) \approx \mathrm{u}^s(x) = 0.25x + 0.5 \tag{1}$$

$$\tanh(x) \approx \mathrm{u}^t(x) = x. \tag{2}$$

Clipping the linear approximations result to,

$$\text{hard-sigmoid}(x) = \max(\min(\mathrm{u}^s(x),\ 1),\ 0) \tag{3}$$

$$\text{hard-tanh}(x) = \max(\min(\mathrm{u}^t(x),\ 1),\ -1). \tag{4}$$

The motivation behind this construction is to introduce linear behavior around zero to allow gradients to flow easily when the unit is not saturated, while providing a crisp decision in the saturated regime.

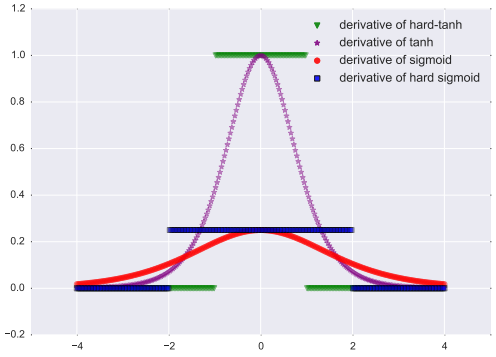

Figure 1. The plot of derivatives of different activation functions.

For the rest of the document we will refer use $h(x)$ to refer to a generic activation function and use $u(x)$ to denote its linearization based on the first-order Taylor expansion about zero. hard-sigmoid saturates when $x \leqslant -2$ or $x \geqslant 2$ and hard-tanh saturates when $x \leqslant -1$ or $x \geqslant 1$. We denote the threshold by x_t. Absolute values of the threshold are $x_t = 2$ for hard-sigmoid and $x_t = 1$ for the hard-tanh.

The ability of the hard-sigmoid and hard-tanh to make crisp decisions comes at the cost of exactly 0 gradients in the saturated regime. This can cause difficulties during training: a small but not infinitesimal change of the pre-activation (before the nonlinearity) may help to reduce the objective function, but this will not be reflected in the gradient.

Remark. Let us note that both $\text{hard-sigmoid}(x)$, $\mathrm{sigmoid}(x)$ and $\tanh(x)$ are all contractive mapping. $\text{hard-tanh}(x)$, becomes a contractive mapping only when

its input is greater than the threshold. An important difference among these activation functions is their fixed points. $\text{hard-sigmoid}(x)$ has a fixed point at $x = \frac{2}{3}$. However the fixed-point of sigmoid is $x = 0$. Any $x \in \mathcal{R}$ between -1 and 1 can be the fixed-point of $\text{hard-tanh}(x)$, but the fixed-point of $\tanh(x)$ is 0. $\tanh(x)$ and $\text{sigmoid}(x)$ have point attractors at their fixed-points. Those mathematical differences among the saturating activation functions can make them behave differently with RNNs and deep networks.

The highly non-smooth gradient descent trajectory may bring the parameters into a state that pushes the activations of a unit towards the 0 gradient regime for a particular example, from where it may become difficult to escape and the unit may get stuck in the 0 gradient regime.

When units saturate and gradients vanish, an algorithm may require many training examples and a lot of computation to recover.

3. Annealing with Noisy Activation Functions

Consider a noisy activation function $\phi(x, \xi)$ in which we have injected iid noise ξ, to replace a saturating nonlinearity such as the hard-sigmoid and hard-tanh introduced in the previous section. In the next section we describe the proposed noisy activation function which has been used for our experiments, but here we want to consider a larger family of such noisy activation functions, when we use a variant of stochastic gradient descent (SGD) for training.

Let ξ have variance σ^2 and mean 0. We want to characterize what happens as we gradually anneal the noise, going from large noise levels ($\sigma \to \infty$) to no noise at all ($\sigma \to 0$).

Furthermore, we will assume that ϕ is such that when the noise level becomes large, so does its derivative with respect to x:

$$\lim_{|\xi| \to \infty} |\frac{\partial \phi(x, \xi)}{\partial x}| \to \infty. \tag{5}$$

In the 0 noise limit, we recover a deterministic nonlinearity, $\phi(x, 0)$, which in our experiments is piecewise linear and allows us to capture the kind of complex function we want to learn. As illustrated in Figure 2, in the large noise limit, large gradients are obtained because backpropagating through ϕ gives rise to large derivatives. Hence, the noise drowns the signal: the example-wise gradient on parameters is much larger than it would have been with $\sigma = 0$. SGD therefore just sees noise and can move around anywhere in parameter space without "seeing" any trend.

Annealing is also related to the signal to noise ratio where SNR can be defined as the ratio of the variance of noise σ_{signal} and σ_{noise}, $SNR = \frac{\sigma_{\text{signal}}}{\sigma_{\text{noise}}}$. If $SNR \to 0$, the model will do pure random exploration. As we anneal SNR will increase, and when σ_{noise} converges to 0, the only source of exploration during the training will come from the noise of Monte Carlo estimates of stochastic gradients.

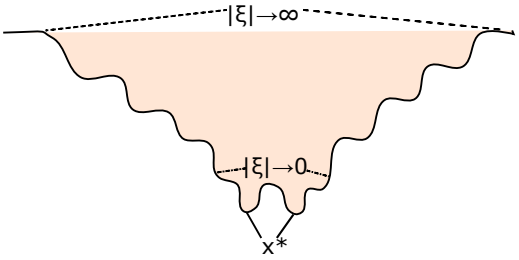

Figure 2. An example of a one-dimensional, non-convex objective function where a simple gradient descent will behave poorly. With large noise $|\xi| \to \infty$, SGD can escape from saddle points and bad local-minima as a result of exploration. As we anneal the noise level $|\xi| \to 0$, SGD will eventually converge to one of the local-minima x^*.

This is precisely what we need for methods such as simulated annealing (Kirkpatrick et al., 1983) and continuation methods (Allgower & Georg, 1980) to be helpful, in the context of the optimization of difficult non-convex objectives. With high noise, SGD is free to explore all parts of space. As the noise level is decreased, it will prefer some regions where the signal is strong enough to be "visible" by SGD: given a finite number of SGD steps, the noise is not averaged out, and the variance continues to dominate. Then as the noise level is reduced SGD spends more time in "globally better" regions of parameter space. As it approaches to zero we are fine-tuning the solution and converging near a minimum of the noise-free objective function. A related approach of adding noise to gradients and annealing the noise was investigated in (Neelakantan et al., 2015) as well. Ge et al. (2015) showed that SGD with annealed noise will globally converge to a local-minima for non-convex objective functions in polynomial number of iterations.

4. Adding Noise Where the Unit Would Saturate

A novel idea behind the proposed noisy activation is that **the noise added to the nonlinearity is proportional to the magnitude of saturation of the nonlinearity**. For $\text{hard-sigmoid}(x)$ and $\text{hard-tanh}(x)$, due to our parametrization of the noise, that translates into the fact that the noise is only added when the hard-nonlinearity saturates. This is different from previous proposals such as the noisy rectifier from Bengio (2013) where noise is added just before a rectifier (ReLU) unit, independently of whether the input is in the linear regime or in the saturating

regime of the nonlinearity.

The motivation is to keep the training signal clean when the unit is in the non-saturating (typically linear) regime and provide some noisy signal when the unit is in the saturating regime.

$\mathrm{h}(x)$ refer to hard saturation activation function such as the hard-sigmoid and hard-tanh introduced in Sec. 2, we consider noisy activation functions of the following form:

$$\phi(x, \xi) = \mathrm{h}(x) + s \tag{6}$$

and $s = \mu + \sigma\xi$. Here ξ is an iid random variable drawn from some generating distribution, and the parameters μ and σ (discussed below) are used to generate a location scale family from ξ.

Intuitively when the unit saturates we pin its output to the threshold value t and add noise. The exact behavior of the method depends on the type of noise ξ and the choice of μ and σ, which we can pick as functions of x in order to let some gradients be propagated even when we are in the saturating regime.

A desirable property we would like ϕ to approximately satisfy is that, in expectation, it is equal to the hard-saturating activation function, i.e.

$$\mathrm{E}_{\xi\sim\mathcal{N}(0,1)}[\phi(x, \xi)] \approx \mathrm{h}(x) \tag{7}$$

If the ξ distribution has zero mean then this property can be satisfied by setting $\mu = 0$, but for biased noise it will be necessary to make other choices for μ. In practice, we used slightly biased ϕ with good results.

Intuitively we would like to add more noise when x is far into the saturated regime, since a large change in parameters would be required desaturate h. Conversely, when x is close to the saturation threshold a small change in parameters would be sufficient for it to escape. To that end we make use of the difference between the original activation function h and its linearization u

$$\Delta = \mathrm{h}(x) - \mathrm{u}(x) \tag{8}$$

when choosing the scale of the noise. See Eqs.1 for definitions of u for the $\mathrm{hard\text{-}sigmoid}$ and $\mathrm{hard\text{-}tanh}$ respectively. The quantity Δ is zero in the unsaturated regime, and when h saturates it grows proportionally to the distance between $|x|$ and the saturation threshold x_t.

We experimented with different ways of scaling σ with Δ, and empirically we found that the following formulation gave good results:

$$\begin{aligned} \sigma(x) &= c\,(\mathrm{g}(p\Delta) - 0.5)^2 \\ \mathrm{g}(x) &= \mathrm{sigmoid}(x). \end{aligned} \tag{9}$$

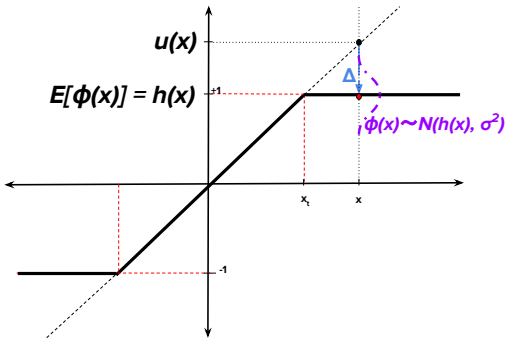

Figure 3. A simple depiction of adding Gaussian noise on the linearized activation function, which brings the average back to the hard-saturating nonlinearity $\mathrm{h}(x)$, in bold. Its linearization is $\mathrm{u}(x)$ and the noisy activation is ϕ. The difference $\mathrm{h}(x) - \mathrm{u}(x)$ is Δ which is a vector indicates the discrepancy between the linearized function and the actual function that the noise is being added to $\mathrm{h}(x)$. Note that, Δ will be zero, at the non-saturating parts of the function where $\mathrm{u}(x)$ and $\mathrm{h}(u)$ matches perfectly.

In Equation 9 a free scalar parameter p is learned during the course of training. By changing p, the model is able to adjust the magnitude of the noise and that also effects the sign of the gradient as well. The hyper-parameter c changes the scale of the standard deviation of the noise.

4.1. Derivatives in the saturated regime

In the simplest case of our method we draw ξ from an unbiased distribution, such as a standard normal. In this case we choose $\mu = 0$ to satisfy Equation 7 and therefore we will have,

$$\phi(x, \xi) = \mathrm{h}(x) + \sigma(x)\xi$$

Due to our parameterization of $\sigma(x)$, when $|x| \leq x_t$ our stochastic activation function behaves exactly as the linear function $\mathrm{u}(x)$, leading to familiar territory. Because Δ will be 0. Let us for the moment restrict our attention to the case when $|x| > x_t$ and h saturates. In this case the derivative of $\mathrm{h}(x)$ is precisely zero, however, if we condition on the sample ξ we have

$$\phi'(x, \xi) = \frac{\partial}{\partial x}\phi(x, \xi) = \sigma'(x)\xi \tag{10}$$

which is non-zero almost surely.

In the non-saturated regime, where $\phi'(x, \xi) = \mathrm{h}'(x)$ the optimization can exploit the linear structure of h near the origin in order to tune its output. In the saturated regime the randomness in ξ drives exploration, and gradients still

flow back to x since the scale of the noise still depends on x. To reiterate, we get gradient information at every point in spite of the saturation of h, and the variance of the gradient information in the saturated regime depends on the variance of $\sigma'(x)\xi$.

4.2. Pushing Activations Towards Linear Regime

An unsatisfying aspect of the formulation with unbiased noise is that, depending on the value of ξ occasionally the gradient of ϕ will point the wrong way. This can cause a backwards message that would push x in a direction that would worsen the objective function on average over ξ. Intuitively we would prefer these messages to "push back" the saturated unit towards a non-saturated state where the gradient of $\mathrm{h}(x)$ can be used safely.

A simple way to achieve this is to make sure that the noise ξ is always positive and adjust its sign manually to match the sign of x. In particular we could set

$$\begin{aligned}\mathrm{d}(x) &= -\operatorname{sgn}(x)\operatorname{sgn}(1-\alpha)\\ s &= \mu(x) + \mathrm{d}(x)\sigma(x)|\xi|.\end{aligned}$$

where ξ and σ are as before and sgn is the sign function, such that $\operatorname{sgn}(x)$ is 1 if x is greater than or equal to 0 otherwise it is -1. We also use the absolute value of ξ in the reparametrization of the noise, such that the noise is being sampled from a half-Normal distribution. We ignored the sign of ξ, such that the direction that the noise pushes the activations are determined by $\mathrm{d}(x)$ and it will point towards $\mathrm{h}(x)$. Matching the sign of the noise to the sign of x would ensure that we avoid the sign cancellation between the noise and the gradient message from backpropagation. $\operatorname{sgn}(1-\alpha)$ is required to push the activations towards $\mathrm{h}(x)$ when the bias from α is introduced.

In practice we use a hyperparameter α that influences the mean of the added term, such that α near 1 approximately satisfies the above condition, as seen in Fig. 4. We can rewrite the noisy term s in a way that the noise can either be added to the linearized function or $\mathrm{h}(x)$. The relationship between Δ, $\mathrm{u}(x)$ and $\mathrm{h}(x)$ is visualized Figure 4.1 can be expressed as in Eqn 11.

We have experimented with different types of noise. Empirically, in terms of performance we found, half-normal and normal noise to be better. In Eqn 11, we provide the formulation for the activation function where $\epsilon = |\xi|$ if the noise is sampled from half-normal distribution, $\epsilon = \xi$ if the noise is sampled from normal distribution.

$$\phi(x,\xi) = \mathrm{u}(x) + \alpha\Delta + \mathrm{d}(x)\sigma(x)\epsilon \tag{11}$$

By using Eqn 11, we arrive at the noisy activations, which we used in our experiments.

$$\phi(x,\xi) = \alpha\mathrm{h}(x) + (1-\alpha)\mathrm{u}(x) + \mathrm{d}(x)\sigma(x)\epsilon \tag{12}$$

As can be seen in Eqn 12, there are three paths that gradients can flow through the neural network, the linear path ($\mathrm{u}(x)$), nonlinear path ($\mathrm{h}(x)$) and the stochastic path ($\sigma(x)$). The flow of gradients through these different pathways across different layers makes the optimization of our activation function easier.

At test time, we used the expectation of Eqn 12 in order to get deterministic units,

$$\mathrm{E}_\xi[\phi(x,\xi)] = \alpha\mathrm{h}(x) + (1-\alpha)\mathrm{u}(x) + \mathrm{d}(x)\sigma(x)\mathrm{E}_\xi[\epsilon] \tag{13}$$

If $\epsilon = \xi$, then $\mathrm{E}_\xi[\epsilon]$ is 0. Otherwise if $\epsilon = |\xi|$, then $\mathrm{E}_\xi[\epsilon]$ is $\sqrt{\frac{2}{\pi}}$.

Algorithm 1 Noisy Activations with Half-Normal Noise for Hard-Saturating Functions

$\Delta \leftarrow h(x) - u(x)$
$\mathrm{d}(x) \leftarrow -\operatorname{sgn}(x)\operatorname{sgn}(1-\alpha)$
$\sigma(x) \leftarrow c\,(g(p\Delta) - 0.5)^2$
$\xi \sim \mathcal{N}(0,\ 1)$
$\phi(x,\xi) \leftarrow \alpha\mathrm{h}(x) + (1-\alpha)\mathrm{u}(x) + (\mathrm{d}(x)\sigma(x)|\xi|)$

To illustrate the effect of α and noisy activation of the $\mathrm{hard\text{-}tanh}$, We provide plots of our stochastic activation functions in Fig 4.

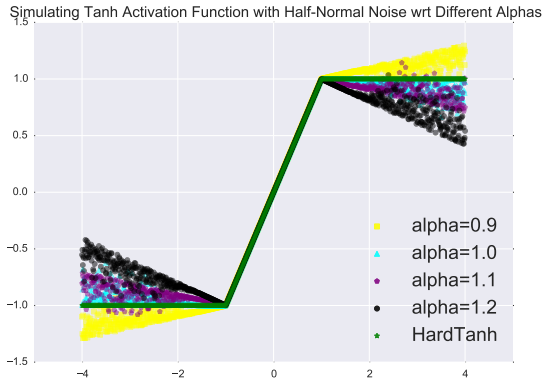

Figure 4. Stochastic behavior of the proposed noisy activation function with different α values and with noise sampled from the Normal distribution, approximating the $\mathrm{hard\text{-}tanh}$ nonlinearity (in bold green).

5. Adding Noise to Input of the Function

Adding noise with fixed standard deviation to the input of the activation function has been investigated for ReLU

activation functions (Nair & Hinton, 2010; Bengio et al., 2013).

$$\phi(x, \xi) = \mathrm{h}(x + \sigma\xi) \text{ and } \xi \sim \mathcal{N}(0, 1). \quad (14)$$

In Eqn 14, we provide a parametrization of the noisy activation function. σ can be either learned as in Eqn 9 or fixed as a hyperparameter.

The condition in Eqn 5 is satisfied only when σ is learned. Experimentally we found small values of σ to work better. When σ is fixed and small, as x gets larger and further away from the threshold x_t, noise will less likely be able to push the activations back to the linear regime. We also investigated the effect of injecting input noise when the activations saturate:

$$\phi(x, \xi) = \mathrm{h}(x + \mathbf{1}_{|x| \geq |x_t|}(\sigma\xi)) \text{ and } \xi \sim \mathcal{N}(0, 1). \quad (15)$$

6. Experimental Results

In our experiments, we used noise only during training: at test time we replaced the noise variable with its expected value. We performed our experiments with just a drop-in replacement of the activation functions in existing experimental setups, without changing the previously set hyperparameters. Hence it is plausible one could obtain better results by performing a careful hyper-parameter tuning for the models with noisy activation functions. In all our experiments, we initialized p uniform randomly from the range $[-1,\ 1]$.

We provide experimental results using noisy activations with normal **(NAN)**, half-normal noise **(NAH)**, normal noise at the input of the function **(NANI)**, normal noise at the input of the function with learned σ **(NANIL)** and normal noise injected to the input of the function when the unit saturates **(NANIS)**. Codes for different types of noisy activation functions can be found at `https://github.com/caglar/noisy_units`.

6.1. Exploratory Analysis

As a sanity-check, we performed small-scale control experiments, in order to observe the behavior of the noisy units. In Fig 5, we showed the learning curves of different types of activations with various types of noise in contrast to the $\tanh$ and hard-$\tanh$ units. The models are single-layer MLPs trained on MNIST for classification and we show the average negative log-likelihood $-\log P(\text{correct class}|\text{input})$. In general, we found that models with noisy activations converge faster than those using $\tanh$ and hard-$\tanh$ activation functions, and to lower NLL than the $\tanh$ network.

We trained $3-$layer MLP on a dataset generated from a mixture of 3 Gaussian distributions with different means and standard deviations. Each layer of the MLP contains 8-hidden units. Both the model with $\tanh$ and noisy$-\tanh$ activations was able to solve this task almost perfectly. By using the learned p values, in Figure 6 and 7, we showed the scatter plot of the activations of each unit at each layer and the derivative function of each unit at each layer with respect to its input.

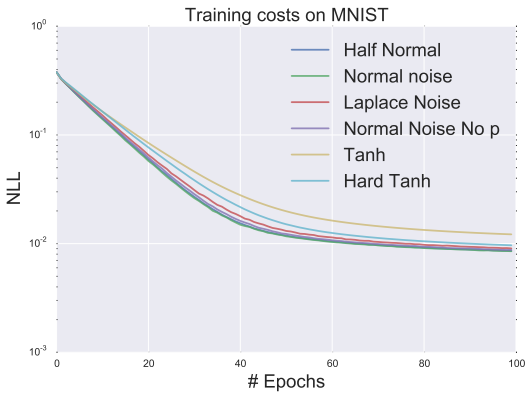

Figure 5. Learning curves of a single layer MLP trained with RMSProp with different noise types and activation functions

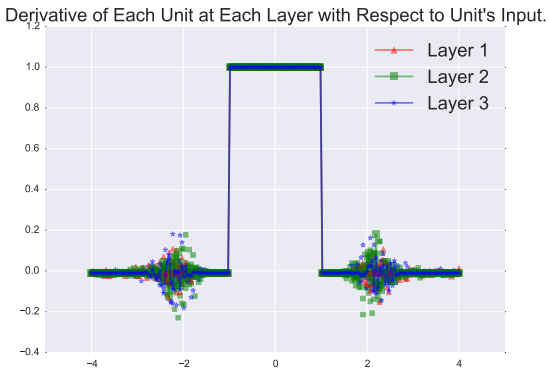

Figure 6. Derivatives of each unit at each layer with respect to its input for a three-layered MLP trained on a dataset generated by three normal distributions with different means and standard deviations. In other words learned $\frac{\partial \phi(x_i^k, \xi_i^k)}{\partial x_i^k}$ at the end of training for i^{th} unit at k^{th} layer. ξ^k is sampled from Normal distribution with $\alpha = 1$.

We further investigated the performances of network with activation functions using NAN, NANI and NANIS on penntreebank (PTB) character-level language modeling. We used a GRU language model over sequences of length 200. We used the same model and train all the activation functions with the same hyperparameters except we ran a grid-search for σ for NANI and NANIS from $[1,\ 0.01]$ with 8

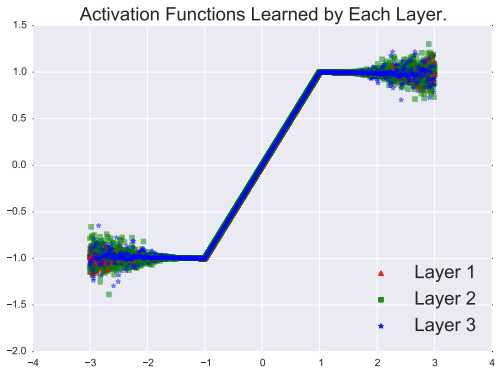

Figure 7. Activations of each unit at each layer of a three-layer MLP trained on a dataset generated by three normal distributions with different means and standard deviations. In other words learned $\phi(x_i^k, \xi_i^k)$ at the end of training for i^{th} unit at k^{th} layer. ξ^k is sampled from Half-Normal distribution with $\alpha = 1$.

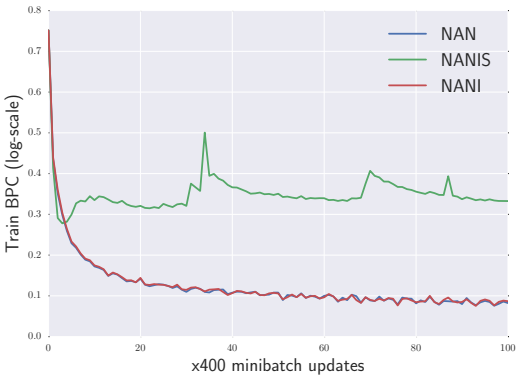

Figure 8. We show the learning curves of a simple character-level GRU language model over the sequences of length 200 on PTB. NANI and NAN, have very similar learning curves. NANIS in the beginning of the training has a better progress than NAN and NANIS, but then training curve stops improving.

values. We choose the best σ based on the validation bit-per-character (BPC). We have not observed important difference among NAN and NANI in terms of training performance as seen on Figure 8.

6.2. Learning to Execute

The problem of predicting the output of a short program introduced in (Zaremba & Sutskever, 2014) [1] proved challenging for modern deep learning architectures. The authors had to use curriculum learning (Bengio et al., 2009) to let the model capture knowledge about the easier examples first and increase the level of difficulty of the examples further down the training.

We replaced all sigmoid and tanh non-linearities in the reference model with their noisy counterparts. We changed the default gradient clipping to 5 from 10 in order to avoid numerical stability problems. When evaluating a network, the length (number of lines) of the executed programs was set to 6 and nesting was set to 3, which are default settings in the released code for these tasks. Both the reference model and the model with noisy activations were trained with "combined" curriculum which is the most sophisticated and the best performing one.

Our results show that applying the proposed activation function leads to better performance than that of the reference model. Moreover it shows that the method is easy to combine with a non-trivial learning curriculum. The results are presented in Table 1 and in Figure 10

[1]The code is residing at https://github.com/wojciechz/learning_to_execute. We thank authors for making it publicly available.

Table 1. Performance of the noisy network on the *Learning to Execute* / task. Just changing the activation function to the proposed noisy one yielded about 2.5% improvement in accuracy.

Model name	Test Accuracy
Reference Model	46.45%
Noisy Network(**NAH**)	**48.09%**

6.3. Penntreebank Experiments

We trained a 2−layer word-level LSTM language model on Penntreebank. We used the same model proposed by Zaremba et al. (2014). [2]We simply replaced all sigmoid and tanh units with noisy hard-sigmoid and hard-tanh units. The reference model is a well-finetuned strong baseline from (Zaremba et al., 2014). For the noisy experiments we used exactly the same setting, but decreased the gradient clipping threshold to 5 from 10. We provide the results of different models in Table 2. In terms of validation and test performance we did not observe big difference between the additive noise from Normal and half-Normal distributions, but there is a substantial improvement due to noise, which makes this result the new state-of-the-art on this task, as far as we know.

6.4. Neural Machine Translation Experiments

We have trained a neural machine translation (NMT) model on the Europarl dataset with the neural attention model (Bahdanau et al., 2014).[3] We have replace all sigmoid and

[2]We used the code provided in https://github.com/wojzaremba/lstm

[3]Again, we have used existing code, provided in https://github.com/kyunghyuncho/dl4mt-material, and

Table 3. Image Caption Generation on Flickr8k. This time we added noisy activations in the code from (Xu et al., 2015) and obtain substantial improvements on the higher-order BLEU scores and the METEOR metric, as well as in NLL. Soft attention and hard attention here refers to using backprop versus REINFORCE when training the attention mechanism. We fixed $\sigma = 0.05$ for NANI and $c = 0.5$ for both NAN and NANIL.

	BLEU -1	BLEU-2	BLEU-3	BLEU-4	METEOR	Test NLL
Soft Attention (Sigmoid and Tanh) (Reference)	**67**	44.8	29.9	19.5	18.9	40.33
Soft Attention (**Noisy Sigmoid and Tanh-NAH**)	66	**45.8**	**30.9**	**20.9**	**20.5**	40.17
Soft Attention (**Noisy Sigmoid and Tanh-NANI**)	66	45.0	30.6	20.7	**20.5**	40.0
Soft Attention (**Noisy Sigmoid and Tanh-NANIL**)	66	44.6	30.1	20.0	**20.5**	**39.9**
Hard Attention (Sigmoid and Tanh)	67	45.7	31.4	21.3	19.5	-

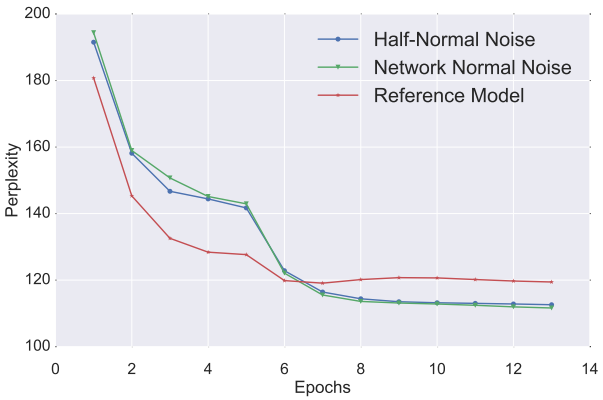

Figure 9. Learning curves of validation perplexity for the LSTM language model on word level on Penntreebank dataset.

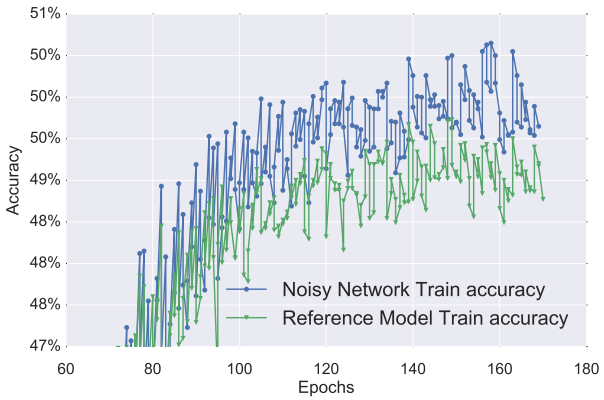

Figure 10. Training curves of the reference model (Zaremba & Sutskever, 2014) and its noisy variant on the "Learning To Execute" problem. The noisy network converges faster and reaches a higher accuracy, showing that the noisy activations help to better optimize for such hard to optimize tasks.

Table 2. Penntreebank word-level comparative perplexities. We only replaced in the code from Zaremba et al. (2014) the sigmoid and tanh by corresponding noisy variants and observe a substantial improvement in perplexity, which makes this the state-of-the-art on this task.

	Valid ppl	Test ppl
Noisy LSTM + **NAN**	111.7	**108.0**
Noisy LSTM + **NAH**	112.6	**108.7**
LSTM (Reference)	119.4	115.6

tanh units in the model with the noisy counterparts. We scaled down the weight matrices initialized to be orthogonal scaled by multiplying with 0.01. Evaluation is done on the `newstest2011` test set. All models are trained with early-stopping. We also compared with a model with hard-tanh and hard-sigmoid units and our model with noisy activations was able to outperform both, as shown in Table 4. Again, we see a substantial improvement (more than 2 BLEU points) with respect to the reference.

only changed the nonlinearities

6.5. Image Caption Generation Experiments

We evaluated our noisy activation functions on a network trained on the Flickr8k dataset. We used the soft neural attention model proposed in (Xu et al., 2015) as our reference model.[4] We scaled down the weight matrices initialized to be orthogonal scaled by multiplying with 0.01. As shown in Table 3, we were able to obtain better results than the reference model and our model also outperformed the best model provided in (Xu et al., 2015) in terms of Meteor score.

6.6. Experiments with Continuation

We performed experiments to validate the effect of annealing the noise to obtain a continuation method for neural networks.

We designed a new task where, given a random sequence of integers, the objective is to predict the number of unique

[4]We used the code provided at `https://github.com/kelvinxu/arctic-captions`.

Table 4. Neural machine Translation on Europarl. Using existing code from (Bahdanau et al., 2014) with nonlinearities replaced by their noisy versions, we find much improved performance (2 BLEU points is considered a lot for machine translation). We also see that simply using the hard versions of the nonlinearities buys about half of the gain.

	Valid nll	BLEU
Sigmoid and Tanh NMT (Reference)	65.26	20.18
Hard-Tanh and Hard-Sigmoid NMT	64.27	21.59
Noisy (**NAH**) Tanh and Sigmoid NMT	**63.46**	**22.57**

Table 5. Experimental Results on the task of finding the unique number of elements in a random integer sequence. This illustrates annealing of the noise level, turning the training procedure into a continuation method. This is combined (bottom) with a curriculum learning strategy, yielding by far the best results.

	Test Error %
LSTM+MLP(Reference)	33.28
Noisy LSTM+MLP(**NAN**)	31.12
Curriculum LSTM+MLP	14.83
Noisy LSTM+MLP(**NAN**) Annealed Noise	**9.53**
Noisy LSTM+MLP(**NANIL**) Annealed Noise	20.94

elements in the sequence. We use an LSTM network over the input sequence, and performed a time average pooling over the hidden states of LSTM to obtain a fixed-size vector. We feed the pooled LSTM representation into a simple (one hidden-layer) ReLU MLP in order to predict the unique number of elements in the input sequence. In the experiments we fixed the length of input sequence to 26 and the input values are between 0 and 10. In order to anneal the noise, we started training with the scale hyperparameter of the standard deviation of noise with $c = 30$ and annealed it down to 0.5 with the schedule of $\frac{c}{\sqrt{t+1}}$ where t is being incremented at every 200 minibatch updates. When noise annealing is combined with a curriculum strategy (starting with short sequences first and gradually increase the length of the training sequences), the best models are obtained.

As a second test, we used the same annealing procedure in order to train a neural turing machine (NTM) on the associative recall task (Graves et al., 2014). We trained our model with a minimum of 2 items and a maximum of 16 items. We show results of the NTM with noisy activations in the controller, with annealed noise, and compare with a regular NTM in terms of validation error. As can be seen in Figure 11, he noisy activation network converges much faster and nails the task, whereas the original network failed to approach a low error.

Figure 11. Validation learning curve of NTM on Associative recall task evaluated over items of length 2 and 16. The NTM with noisy controller converges much faster and solves the task.

7. Conclusion

Nonlinearities in neural networks are both a blessing and a curse. A blessing because they allow to represent more complicated functions and a curse because that makes the optimization more difficult. For example, we have found in our experiments that using a hard version (hence more nonlinear) of the sigmoid and $\tanh$ nonlinearities often improved results. In the past, various strategies have been proposed to help deal with the difficult optimization problem involved in training some deep networks, including curriculum learning, which is an approximate form of continuation method. Earlier work also included softened versions of the nonlinearities that are gradually made harder during training. Motivated by this prior work, we introduce and formalize the concept of noisy activations as a general framework for injecting noise in nonlinear functions so that large noise allows SGD to be more exploratory. We propose to inject the noise to the activation functions either at the input of the function or at the output where unit would otherwise saturate, and allow gradients to flow even in that case. We show that our noisy activation functions are easier to optimize. It also, achieves better test errors, since the noise injected to the activations also regularizes the model as well. Even with a fixed noise level, we found the proposed noisy activations to outperform their sigmoid or $\tanh$ counterpart on different tasks and datasets, yielding state-of-the-art or competitive results with a simple modification, for example on PennTreebank. In addition, we found that annealing the noise to obtain a continuation method could further improved performance.

References

Allgower, E. L. and Georg, K. *Numerical Continuation Methods. An Introduction*. Springer-Verlag, 1980.

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

Bengio, Yoshua. Estimating or propagating gradients through stochastic neurons. Technical Report arXiv:1305.2982, Universite de Montreal, 2013.

Bengio, Yoshua, Louradour, Jerome, Collobert, Ronan, and Weston, Jason. Curriculum learning. In *ICML'09*, 2009.

Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

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

Ge, Rong, Huang, Furong, Jin, Chi, and Yuan, Yang. Escaping from saddle points—online stochastic gradient for tensor decomposition. *arXiv preprint arXiv:1503.02101*, 2015.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Hermann, Karl Moritz, Kocisky, Tomas, Grefenstette, Edward, Espeholt, Lasse, Kay, Will, Suleyman, Mustafa, and Blunsom, Phil. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pp. 1684–1692, 2015.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Karpathy, Andrej, Johnson, Justin, and Fei-Fei, Li. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Kirkpatrick, S., Jr., C. D. Gelatt, , and Vecchi, M. P. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521(7553):436–444, 2015.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

Neelakantan, Arvind, Vilnis, Luke, Le, Quoc V, Sutskever, Ilya, Kaiser, Lukasz, Kurach, Karol, and Martens, James. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Courville, Aaron, Salakhutdinov, Ruslan, Zemel, Richard, and Bengio, Yoshua. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

Zaremba, Wojciech and Sutskever, Ilya. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

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

Acknowledgements

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, Samsung, the Canada Research Chairs and CIFAR. We would also like to thank the developers of Theano [5], for developing such a powerful tool for scientific computing. Caglar Gulcehre also thanks to IBM Watson Research and Statistical Knowledge Discovery Group at IBM Research for supporting this work during his internship.

[5] http://deeplearning.net/software/theano/