A Probabilistic Framework for Nonlinearities in Stochastic Neural Networks

Authored by:

Lawrence Carin
Qinliang Su
xuejun Liao

Abstract

We present a probabilistic framework for nonlinearities, based on doubly truncated Gaussian distributions. By setting the truncation points appropriately, we are able to generate various types of nonlinearities within a unified framework, including sigmoid, tanh and ReLU, the most commonly used nonlinearities in neural networks. The framework readily integrates into existing stochastic neural networks (with hidden units characterized as random variables), allowing one for the first time to learn the nonlinearities alongside model weights in these networks. Extensive experiments demonstrate the performance improvements brought about by the proposed framework when integrated with the restricted Boltzmann machine (RBM), temporal RBM and the truncated Gaussian graphical model (TGGM).

1 Paper Body

A typical neural network is composed of nonlinear units connected by linear weights, and such a network is known to have universal approximation ability under mild conditions about the nonlinearity used at each unit [1, 2]. In previous work, the choice of nonlinearity has commonly been taken as a part of network design rather than network learning, and the training algorithms for neural networks have been mostly concerned with learning the linear weights. However, it is becoming increasingly understood that the choice of nonlinearity plays an important role in model performance. For example, [3] showed advantages of rectified linear units (ReLU) over sigmoidal units in using the restricted Boltzmann machine (RBM) [4] to pre-train feedforward ReLU networks. It was further shown in [5] that rectified linear units (ReLU) outperform sigmoidal units in a generative network under the same undirected and bipartite structure as the RBM. A number of recent works have reported benefits of learning nonlinear units along with the inter-unit weights. These methods are based on using parameterized nonlinear functions to activate each unit in a neural network, with the unit-dependent parameters incorporated into the data-driven training algorithms. In particular, [6] considered the adaptive piecewise linear (APL) unit defined by a mixture of hinge-shaped functions, and [7] used non-parametric Fourier basis expansion to construct the activation function of each unit. The maxout network [8] employs piecewise linear convex (PLC) units, where each PLC unit is obtained by max-pooling over multiple linear units. The PLC units were extended to Lp units in [9] where the normalized Lp norm of multiple linear units yields the output of an Lp unit. All these methods have been developed for learning the deterministic characteristics of a unit, lacking a stochastic unit characterization. The deterministic nature limits these methods from being easily applied to stochastic neural networks (for which the hidden units are random variables, rather than being characterized by a deterministic function), such as Boltzmann machines [10], restricted Boltzmann machines [11], and sigmoid belief networks (SBNs) [12]. We propose a probabilistic framework to unify the sigmoid, hyperbolic tangent (tanh) and ReLU nonlinearities, most commonly used in neural networks. The proposed framework represents a 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

unit h probabilistically as p(h—z, ?), where z is the total net contribution that h receives from other units, and ? represents the learnable parameters. By taking the expectation of h, a deterministic R characterization of the unit is obtained as E(h—z, ?) , h p(h—z, ?)dh. We show that the sigmoid, tanh and ReLU are well approximated by E(h—z, ?) under appropriate settings of ?. This is different from [13], in which nonlinearities were induced by the additive noises of different variances, making the model learning much more expensive and nonlinearity producing less flexible. Additionally, more-general nonlinearities may be constituted or learned, with these corresponding to distinct settings of ?. A neural unit represented by the proposed framework is named a truncated Gaussian (TruG) unit because the framework is built upon truncated Gaussian distributions. Because of the inherent stochasticity, TruG is particularly useful in constructing stochastic neural networks. The TruG generalizes the probabilistic ReLU in [14, 5] to a family of stochastic nonlinearities, with which one can perform two tasks that could not be done previously: (i) One can interchangeably use one nonlinearity in place of another under the same network structure, as long as they are both in the TruG family; for example, the ReLU-based stochastic networks in [14, 5] can be extended to new networks based on probabilistic tanh or sigmoid nonlinearities, and the respective algorithms in [14, 5] can be employed to train the associated new models with little modification; (ii) Any stochastic network constructed with the TruG can learn the nonlinearity alongside the network weights, by maximizing the likelihood function of ? given the training data. We can learn the nonlinearity at the unit level, with each TruG unit having its own parameters; or we can learn the nonlinearity at the model level, with the entire network sharing the same parameters for all its TruG units. The different choices entail only minor changes in the update equation of ?, as will be seen subsequently. We integrate the

TruG framework into three existing stochastic networks: the RBM, temporal RBM [15] and feedforward TGGM [14], leading to three new models referred to as TruG-RBM, temporal TruG-RBM and TruG-TGGM, respectively. These new models are evaluated against the original models in extensive experiments to assess the performance gains brought about by the TruG. To conserve space, all propositions in this paper are proven in the Supplementary Material.

2

TruG: A Probabilistic Framework for Nonlinearities in Neural Networks

For a unit h that receives net contribution z from other units, we propose to relate h to z through the following stochastic nonlinearity,

N h z, ? 2 I(?1 ? h ? ?2) (1) , N[?1 ,?2] h z, ? 2 , p(h—z, ?) = R ?2 0 2 0 N (h —z, ?) dh ?1

where I(?) is an indicator function and N ? z, ? 2 is the probability density function (PDF) of a univariate Gaussian distribution with mean z and variance ? 2 ; the shorthand notation N[?1 ,?2] indicates the density N is truncated and renormalized such that it is nonzero only in the interval [?1 , ?2]; ? , {?1 , ?2 } contains the truncation points and ? 2 is fixed. The units of a stochastic neural network fall into two categories: visible units and hidden units [4]. The network represents a joint distribution over both hidden and visible units and the hidden units are integrated out to yield the marginal distribution of visible units. With a hidden unit expressed in (1), the expectation of h is given by E(h—z, ?) = z + ?

?(?1??z) ? ?(?2??z) ?(?2??z) ? ?(?1??z)

,

(2)

where ?(?) and ?(?) are, respectively, the PDF and cumulative distribution function (CDF) of the standard normal distribution [16]. As will become clear below, a weighted sum of these expected hidden units constitutes the net contribution received by each visible unit when the hidden units are marginalized out. Therefore E(h—z, ?) acts as a nonlinear activation function to map the incoming contribution h receives to the outgoing contribution h sends out. The incoming contribution received by h may be a random variable or a function of data such as z = wT x + b; the former case is typically for unsupervised learning and the latter case for supervised learning with x being the predictors. By setting the truncation points to different values, we are able to realize many different kinds of nonlinearities. We plot in Figure 1 three realizations of E(h—z, ?) as a function of z, each with a particular setting of {?1 , ?2 } and ? 2 = 0.2 in all cases. The plots of ReLU, tanh and sigmoid are 2

1

10 5 0 -20

-10

0

10

20

0.5

0.8

0 -0.5 -1 -20
6
sigmoid TruG
0.6 0.4 0.2
-10
0
z
10
20
0 -20
4 3 2 1
-10
0
10
20
0 -20
-10
0
z
z
(a)
sigmoid TruG ReLU
5
activation(z)
15
1
tanh TruG
activation(z)
ReLU TruG
activation(z)
activation(z)
20
(b)
10
20
z
(c)
(d)

Figure 1: Illustration of different nonlinearities realized by the TruG with different truncation points. (a) ?1 = 0 and ?2 = +?; (b) ?1 = ?1 and ?2 = 1; (c) ?1 = 0 and ?2 = 1; (d) ?1 = 0 and ?2 = 4. also shown as a comparison. It is seen from Figure 1 that, by choosing appropriate truncation points, E(h—z, ?) is able to approximate ReLU, tanh and sigmoid, the three types of nonlinearities most widely used in neural networks. We can also realize other types of nonlinearities by setting the truncation points to other values, as exemplified in Figure 1(d). The truncation points can be set manually by hand, selected by cross-validation,

or learned in the same way as the inter-unit weights. In this paper, we focus on learning them alongside the weights based on training data. The variance of h, given by [16],

Var(h—z, ?) = ? 2 +

?1 ?z ?1 ?z ? ? ?

?2 ?2 ?z ? ?

?

?2 ?z ? ?

??

?1 ?z ?

?2 ?z ?

? ? ?2 ?

?

?1 ?z ?

??

?

?2 ?z ?

??

?2 ?z ? ?1 ?z ?

?2 ? , (3)

is employed in learning the truncation points and network weights. Direct evaluation of (2) and (3) is prone to the numerical issue of 00 , because both ?(z) and ?(z) are so close to 0 when z ¡ ?38 that they are beyond the maximal accuracy a double float number can represent. We solve this problem by ?(z) using the fact that (2) and (3) can be equivalently expressed in terms of ?(z) by dividing both the numerator and the denominator by ?(?). We make use of the following approximation for the ratio, ? ?(z) z2 + 4 ? z ? , ?(z), for z ¡ ?38, (4) ?(z) 2 the accuracy of which is established in Proposition 1.

? 2 +4?z

?(z) ? 1 ¡ 2 ?zz2 +8?3z ?1; moreover, for all Proposition 1. The relative error is bounded by ?(z)/ ?(z)

?(z) z ¡ ?38, the relative error is guaranteed to be smaller than 4.8 ? 10?7 , that is, ?(z)/ ?(z) ? 1 ¡ 4.8 ? 10?7 for all z ¡ ?38.

3

RBM with TruG Nonlinearity

We generalize the ReLU-based RBM in [5] by using the TruG nonlinearity. The resulting TruG-RBM is defined by the following joint distribution over visible units x and hidden units h, p(x, h) =

1 ?E(x,h) e I(x ? {0, 1}n , ?1 ? h ? ?2), Z

(5)

where E(x, h) , 12 hT diag(d)h ? xT Wh ? bT x ? cT h is an energy function and Z is the normalization constant. Proposition 2 shows (5) is a valid probability distribution. Proposition 2. The distribution p(x, h) defined in (5) is normalizable. Qn By (5), the conditional distribution of x given h is still Bernoulli, p(x—h) = i=1 ?([Wh + b]i), while the conditional p(h—x) is a truncated normal distribution, i.e.,

m Y 1 1 T
p(h—x) = N[?1 ,?2] hj [W x + c]j , . (6) dj dj j=1 3
By setting ?1 and ?2 to different values, we are able to produce different nonlinearities in (6). We train a TruG-RBM based on maximizing the log-likelihood function ‘(?, ?) , P ln p(x; ?, ?), where ? , {W, b, c} denotes the network weights, p(x; ?, ?) , R ?2x?X p(x, h)dh is contributed by a single data point x, and X is the training dataset. ?1 3.1
The Gradient w.r.t. Network Weights
The gradient w.r.t. ? is known to be
?ln p(x) ??
=E
h
?E(x,h) ??
i
?E
h
i , where E[?] and E[?—x]
?E(x,h) ?? x
means the expectation w.r.t. p(x, h) and p(h—x), respectively. If we estimate the gradient using a standard sampling-based method, the variance associated with the estimate is usually very large. To reduce the variance, we follow the traditional RBM in applying the contrastive divergence (CD) to estimate the gradient [4]. Specifically, we approximate the gradient as
? ln p(x) ?E(x, h) ?E(x, h) (k) x ?E x , (7) ?E ?? ?? ?? where x(k) is the k-th sample of the Gibbs sampler p(h(1) —x(0)), p(x(1) —h(1)) ? ? ? p(x(k) —h(k)), with x(0) being the data x. As shown in (6), p(x—h) and p(h—x) are factorized Bernoulli and univariate truncated normal distributions, for which efficient sampling algorithms exist [17, 18]. = xi , ?E(x,h) = hj and ?E(x,h) = 12 h2j . ?cj
2 (s) ?dj (s) Thus estimation of the gradient with CD only requires E hj —x and E hj —x , which can be calculated using (2) and (3). Using the estimated gradient, the weights can be updated using the stochastic gradient ascent algorithm or its variants. Furthermore, we can obtain that
3.2
?E(x,h) ?wij
= xi hj ,
?E(x,h) ?bi
The Gradient w.r.t. Truncation Points
The gradient w.r.t. ?1 and ?2 are given by m
? ln p(x) X = (p(hj = ?1) ? p(hj = ?1 —x)) , ??1 j=1
(8)
m
? ln p(x) X = (p(hj = ?2 —x) ? p(hj = ?2)) , ??2 j=1
(9)
for a single data point, with provided inthe Supplementary Material. It is known that

the derivation 1 T p(hj = ?—x) = N[?1 ,?2] hj = ? dj [W x + c]j , d1j , which can be easily calculated. However, if we calculate p(hj = ?) directly, P it would be computationally prohibitive. Fortunately, by noticing the identity p(hj = ?) = x p(hj = ?—x)p(x), are able to estimate it efficiently with CD as we

[WT x(k)+c]j 1 (k) p(hj = ?) ? p(hj = ?—x) = N[?1 ,?2] hj = ? , dj , where x(k) is the k-th sample of dj the Gibbs sampler as described above. Therefore, the gradient w.r.t. the lower and upper truncation

Pm points can be estimated using the equations ? ln??p(x) ? j=1 p(hj = ?2 —x)?p(hj = ?2 —x(k)) and 2

Pm ? ln p(x) ? ? j=1 p(hj = ?1 —x)?p(hj = ?1 —x(k)) . After obtaining the gradients, we can update the ??1 truncation points with stochastic gradient ascent methods. It should be emphasized that in the derivation above, we assume a common truncation point pair {?1 , ?2 } shared among all units for the clarity of presentation. The extension to separate truncation points for different units is straightforward, by simply replacing (8) and (9) with ? ln p(x) p(x) = (p(hj = ?2j —x) ? p(hj = ?2j)) and ? ln = (p(hj = ?1j) ? p(hj = ?1j —x)), where ??2j ??1j ?1j and ?2j are the lower and upper truncation point of j-th unit, respectively. For the models discussed subsequently, one can similarly get the gradient w.r.t. unit-dependent truncations points. After training, due to the conditional independence between x and h and the existence of efficient sampling algorithm for truncated normal, samples can be drawn efficiently from the TruG-RBM using the Gibbs sampler discussed below (7). 4

4

Temporal RBM with TruG Nonlinearity

We integrate the TruG framework into the temporal RBM (TRBM) [19] to learn the probabilistic nonlinearity in sequential-data modeling. The resulting temporal TruG-RBM is defined by QT p(X, H) = p(x1 , h1) t=2 p(xt , ht —xt?1 , ht?1), (10) where p(x1 , h1) and p(xt , ht —xt?1 , ht?1) are both represented by TruG-RBMs; xt ? Rn and ht ? Rm are the visible and hidden variables at time step t, with X , [x1 , x2 , ? ? ? , xT] and H , [h1 , h2 , ? ? ? , hT]. To be specific, the distribution p(xt , ht —xt?1 , ht?1) is defined as p(xt , ht —xt?1 , ht?1) = Z1t exp?E(xt ,ht) I(x ? {0, 1}n , ?1 ? ht ? ?2),

where the energy function takes the form E(xt , ht) , 12 xTt diag(a) xt + hTt diag(d) ht ?

T T 2xTt W1 ht ? 2cT ht ? 2 (W2 xt?1) ht ? 2bT xt ?2 (W3 xt?1) xt ? 2(W4 ht?1)T ht ; and R +? R +? Zt , ?? 0 e?E(xt ,ht) dht dxt . Similar to the TRBM, directly optimizing the log-likelihood is difficult. We instead optimize the lower bound L , Eq(H—X) [ln p(X, H; ?, ?) ? ln q(H—X)] ,

(11)

where q(H—X) is an approximating posterior distribution of H. The lower bound is equal to the log-likelihood when q(H—X) is exactly the true posterior p(H—X). We follow [19] to choose the following approximate posterior, q(H—X) = p(h1 —x1) ? ? ? p(hT —xT ?1 , hT ?1 , xT), with which it can be shown that the gradient of the lower bound w.r.t. h the network i

PT ?E(xt ,ht) ?L weights is given by ?? = ? E E p(h —x ,h ,x) p(x ,h

—x ,h) t?1 t?2 t?2 t?1 t t t?1 t?1 ?? h i t=1 ?E(xt ,ht) Ep(ht —xt?1 ,ht?1 ,xt) . At any time step t, the outside expectation (which is over ht?1) is ?? approximated by sampling from p(ht?1 —xt?2 , ht?2 , xt?1); given ht?1 and xt?1 , one can represent p(xt , ht —xt?1 , ht?1) as a TruG-RBM and therefore the two inside expectations can be computed in the same way as in Section 3. In particular, the variables in ht are conditionally independent given Qm (xt?1 , ht?1 , xt), i.e., p(ht —xt?1 , ht?1 , xt) = j=1 p(hjt —xt?1 , ht?1 , xt) with each component equal to

[W1T xt+W2 xt?1+W4 ht?1+c]j 1 p(hjt —xt?1 , ht?1 , xt) =N[?1 ,?2] hjt , . (12) dj dj Similarly, the variables in xt are conditionally independent given (xt?1 , ht?1 , ht). As a result, Ep(ht —xt?1 ,ht?1 ,xt) [?] can be calculated in closed-form using (2) and (3), and Ep(xt ,ht —xt?1 ,ht?1 ,xt) [?] can be estimated using the CD algorithm, as in Section Section 3. The gradient of L w.r.t. the upper truncation point is X

T X m T X m X ?L = Eq(H—X) p(hjt = ?2 —xt?1 , ht?1 , xt) ? p(hjt = ?2 —xt?1 , ht?1) , ??2 t=1 j=1 t=1 j=1 ?L with ?? taking a similar form, where the expectations are similarly calculated using the same 1 ?L approach as described above for ?? .

5

TGGM with TruG Nonlinearity

We generalize the feedforward TGGM model in [14] by replacing the probabilistic ReLU with the TruG. The resulting TruG-TGGM model is defined by the joint PDF over visible variables y and hidden variables h, p(y, h—x) = N (y—W1 h + b1 , ? 2 I)N[?1 ,?2] (h—W0 x + b0 , ? 2 I), 5

(13)

given the predictor variables x. After marginalizing out h, we get the expectation of y as E[y—x] = W1 E(h—W0 x + b0 , ?) + b1 ,

(14)

where E(h—W0 x + b0 , ?) is given element-wisely in (2). It is then clear that the expectation of y is related to x through the TruG nonlinearity. Thus E[y—x] yields the same output as a three-layer perceptron that uses (2) to activate its hidden units. Hence, the TruG-TGGM model defined in (13) can be understood as a stochastic perceptron with the TruG nonlinearity. By choosing different values for the truncation points, we are able to realize different kinds of nonlinearities, including ReLU, sigmoid and tanh. ToRtrain the model by maximum likelihood estimation, we need to know the gradient of ln p(y—x) , ln p(y, h—x; ?)dh, where ? , {W1 , W0 , b1 , b0 } represents the model parameters. By rewriting ?E(y,h,x) the joint PDFh as p(y, h—x) to be given by I(? i ? h e i 1 ? h ? ?2), the gradient is found ?E(y,h,x) ?E(y,h,x) ——y?W1 h?b1 ——2 +——h?W0 x?b0 ——2 ? ln p(y—x) =E ?? x ?E ?? x, y , where E(y, h, x) , ; ?? 2? 2 E[?—x] is the expectation w.r.t. p(y, h—x); and E[?—x, y] is the expectation w.r.t. p(h—x, y). From (13), we know p(h—x) = N[?1 ,?2] (h—W0 x + b0 , ? 2 I) can be factorized into a product of univariate truncated Gaussian PDFs. Thus the expectation E[h—x] can be computed using (2). However, the expectations E[h—x, y] and E[hhT —x, y] involve a multivariate truncated Gaussian PDF and are expensive to calculate directly. Hence mean-field variational Bayesian analysis is used to compute the approximate expectations. The details are similar to those in [14] except that (2) and (3) are used to calculate the expectation and variance of h. p(y—x) = The gradients of the log-likelihood w.r.t. the truncation points ?1 and ?2 are given by ? ln?? 2 PK PK ? ln p(y—x) = ? j=1 (p(hj = ?1 —y, x) ? p(hj = ?1 —x)) j=1 (p(hj = ?2 —y, x) ? p(hj = ?2 —x)) and ??1 for a single data point, with the derivation provided in the Supplementary Material. The probability p(hj = ?1 —x) can be computed directly since it is a univariate truncated Gaussian distribution. For p(hj = ?2 —y, x), we approximate it with the mean-field marginal distributions obtained above.

Although TruG-TGGM involves random variables, thanks to the existence of close-form expression ?, for the expectation of univariate truncated normal, the testing is still very easy. Given a predictor x the output can be simply predicted with the conditional expectation E[y—x] in (14).

6

Experimental Results

We evaluate the performance benefit brought about by the TruG framework when integrated into the RBM, temporal RBM and TGGM. In each of the three cases, the evaluation is based on comparing the original network to the associated new network with the TruG nonlinearity. For the TruG, we either manually set {?1 , ?2 } to particular values, or learn them automatically from data. We consider both the case of learning a common {?1 , ?2 } shared for all hidden units and the case of learning a separate {?1 , ?2 } for each hidden unit. Results of TruG-RBM The binarized Table 1: Averaged test log-probability on MNIST. (?) MNIST and Caltech101 Silhouettes are con- Results reported in [20]; () Results reported in [21] sidered in this experiment. The MNIST using RMSprop as the optimizer. contains 60,000 training and 10,000 testing Ave. Log-prob Model Trun. Points images of hand-written digits, while CalMNIST Caltech101 tech101 Silhouettes includes 6364 training [0, 1] -97.3 -127.9 and 2307 testing images of objects? silhou[0, +?) -83.2 -105.2 ettes. For both datasets, each image has TruG-RBM [-1, 1] -124.5 -141.5 28 ? 28 pixels [22]. Throughout this experc-Learn -82.9 -104.6 iment, 500 hidden units are used. RMSprop s-Learn -82.5 -104.3 is used to update the parameters, with the RBM ? -86.3? -109.0 delay and mini-batch size set to 0.95 and 100, respectively. The weight parameters are initialized with the Gaussian noise of zero mean and 0.01 variance, while the lower and upper truncation points at all units are initialized to 0 and 1, respectively. The learning rates for weight parameters are fixed to 10?4 . Since truncations points influence the whole networks in a more fundamental way than weight parameters, it is observed that smaller learning rates are often preferred for them. To balance the convergence speed and 6

Sigmoid function ¡(7) Nonlinearity in Ball Nonlinearity in MNIST Nonlinearity in Motion Nonlinearity in Caltech

0.12

0.1

2.5 2

1.5

Probability
0.1
Probability
Output after transform
3
0.12
0.14
4 3.5
0.08 0.06
0.08 0.06 0.04
0.04
1
0.02 0.02
0.5
0 0 -15
-10
-5
0
5
Input before transform: 7
(a)
10
15
0.5
1
1.5
2
2.5
Upper truncation point:
(b)
3 2
3.5
2
2.5
3
3.5
Upper truncation point:
4
4.5
2
(c)

Figure 2: (a) The learned nonlinearities in TruG models with shared upper truncation point ?; The distribution of unit-level upper truncation points of TruG-RBM for (b) MNIST; (c) Caltech101 Silhouettes. performance, we anneal their learning rates from 10?4 to 10?6 gradually. The evaluation is based on the log-probability averaged over test data points, which are estimated using

annealed importance sampling (AIS) [23] with 5 ? 105 inverse temperatures equally spaced in [0, 1]; the reported test log-probability is averaged over 100 independent AIS runs. To investigate the impact of truncation points, we first set the lower and upper truncation points to three fixed pairs: [0, 1], [0, +?) and [?1, 1], which correspond to probabilistic approximations of sigmoid, ReLU and tanh nonlinearities, respectively. From Table 1, we see that the ReLU-type TruG-RBM performs much better than the other two types of TruG-RBM. We also learn the truncation points from data automatically. We can see that the model benefits significantly from nonlinearity learning, and the best performance is achieved when the units learn their own nonlinearities. The learned common nonlinearities (c-Learn) for different datasets are plotted in Figure 2(a), which shows that the model always tends to choose a nonlinearity in between sigmoid and ReLU functions. For the case with separate nonlinearities (s-Learn), the distributions of the upper truncation points in the TruGRBM?s for MNIST and Caltech101 Silhouettes are plotted in Figure 2(b) and (c), respectively. Note that due to the detrimental effect observed for negative truncation points, here the lower truncation points are fixed to zero and only the upper points are learned. To demonstrate the reliability of AIS estimate, the convergence plots of estimated log-probabilities are provided in Supplementary Material. Results of Temporal TruG-RBM The Bouncing Ball and CMU Motion Capture datasets are considered in the experiment with temporal models. Bouncing Ball consists of synthetic binary videos of 3 bouncing balls in a box, with 4000 videos for training and 200 for testing, and each video has 100 frames of size 30 ? 30. CMU Motion Capture is composed of data samples describing the joint angles associated with different motion types. We follow [24] to train a model on 31 sequences and test the model on two testing sequences (one is running and the other is walking). Both the original TRBM and the TruG-TRBM use 400 hidden units for Bouncing Ball and 300 hidden units for CMU Motion Capture. Stochastic gradient descent (SGD) is used to update the parameters, with the momentum set to 0.9. The learning rates are set to be 10?2 and 10?4 for the two datasets, respectively. The learning rate for truncation points is annealed gradually, as done in Section 6. Since calculating the log-probabilities for these temporal models is computationally prohibitive, prediction error is employed here as the performance evaluation criteria, which is widely used [24, 25] in temporal generative models. The performances averaged over 20 independent runs are reported here. Tables 2 and 3 confirm again that models benefit remarkably from nonlinearity learning, especially in the case of learning a separate nonlinearity for each hidden unit. It is noticed that, although the ReLU-type TruG-TRBM performs better the tanh-type TruG-TRBM on Bouncing Ball, the former performs much worse than the latter on CMU Motion Capture. This demonstrates that a fixed nonlinearity cannot perform well on every dataset. However, by learning truncation points automatically, the TruG can adapt the nonlinearity to the data and thus performs the best on every dataset (up to the representational limit of the TruG framework). Video samples drawn from the trained models are provided in the Supplementary Material. Results of TruG-TGGM Ten datasets from the UCI repository are used in this experiment. Following the procedures in [26], datasets are randomly partitioned into training and testing subsets for 7

Table 2: Test prediction error on Bouncing Ball. (?) Taken from [24], in which 2500 hidden units are used. Model TruG-TRBM TRBM RTRBM?

Trun. Points [0, 1] [0, +?) [-1, 1] c-Learn s-Learn ? ?

Pred. Err. 6.38?0.51 4.16?0.42 6.01?0.52 3.82?0.41 3.66?0.46 4.90?0.47 4.00?0.35

Table 3: Test prediction error on CMU Motion Capture, in which ?w? and ?r? mean walking and running, respectively. (?) Taken from [24]. Model TruG-TRBM TRBM ss-SRTRBM?

Trun. Points [0, 1] [0, +?) [-1, 1] c-Learn s-Learn ? ?

Err. (w) 8.2?0.18 21.8?0.31 7.3?0.21 6.7?0.29 6.8?0.24 9.6?0.15 8.1?0.06

Err. (r) 6.1?0.22 14.9?0.29 5.9?0.22 5.5?0.22 5.4?0.14 6.8?0.12 5.9?0.05

Table 4: Averaged test RMSEs for multilayer perception (MLP) and TruG-TGGMs under different truncation points. (?) Results reported in [26], where BH, CS, EE, K8 NP, CPP, PS, WQR, YH, YPM are the abbreviations of Boston Housing, Concrete Strength, Kin8nm, Naval Propulsion, Cycle Power Plant, Protein Structure, Wine Quality Red, Yacht Hydrodynamic, Year Prediction MSD, respectively. Dataset

MLP (ReLU)?

BH CS EE K8 NP CPP PS WQR YH YPM

3.228 ?0.195 5.977?0.093 1.098?0.074 0.091?0.002 0.001?0.000 4.182?0.040 4.539?0.029 0.645?0.010 1.182?0.165 8.932?N/A

TruG-TGGM with Different Trun. Points [0, 1] 3.564?0.655 5.210?0.514 1.168?0.130 0.094?0.003 0.002?0.000 4.023?0.128 4.231?0.083 0.662?0.052 0.871?0.367 8.961?N/A

[0, +?) 3.214?0.555 5.106?0.573 1.252?0.123 0.086?0.003 0.002?0.000 4.067?0.129 4.387?0.072 0.644?0.048 0.821?0.276 8.985?N/A

[-1, 1] 4.003?0.520 4.977?0.482 1.069?0.166 0.091?0.003 0.002? 0.000 3.978?0.132 4.262?0.079 0.659?0.052 0.846?0.310 8.859?N/A

c-Learn 3.401?0.375 4.910?0.467 0.881?0.079 0.073?0.002 0.001?0.000 3.952?0.134 4.209?0.073 0.645?0.050 0.803?0.292 8.893?N/A

s-Learn 3.622? 0.538 4.743? 0.571 0.913? 0.120 0.075? 0.002 0.001? 0.000 3.951? 0.130 4.206? 0.071 0.643? 0.048 0.793? 0.289 8.965? N/A

10 trials except the largest one (Year Prediction MSD), for which only one partition is conducted due to computational complexity. Table 4 summarizes the root mean square error (RMSE) averaged over the different trials. Throughout the experiment, 100 hidden units are used for the two datasets (Protein Structure and Year Prediction MSD), while 50 units are used for the remaining. RMSprop is used to optimize the parameters, with RMSprop delay set to 0.9. The learning rate is chosen from the set {10?3 , 2 ? 104 , 10?4 }, while the mini-batch size is set to 100 for the two largest datasets and 50 for the others. The number of VB cycles used in the inference is set to 10 for all datasets. The RMSE?s of TGGMs with fixed and learned truncation points are reported in Table 4, along with the RMSE?s of the (deterministic) multilayer perceptron (MLP) using ReLU nonlinearity for comparison. Similar to what we

have observed in generative models, the supervised models also benefit significantly from nonlinearity learning. The TruG-TGGM with learned truncation points perform the best for most datasets, with the separate learning performing slightly better than the common learning overall. Due to the limited space, the learned nonlinearities and their corresponding truncation points are provided in Supplementary Material.

7

Conclusions

We have presented a probabilistic framework, termed TruG, to unify ReLU, sigmoid and tanh, the most commonly used nonlinearities in neural networks. The TruG is a family of nonlinearities constructed with doubly truncated Gaussian distributions. The ReLU, sigmoid and tanh are three important members of the TruG family, and other members can be obtained easily by adjusting the lower and upper truncation points. A big advantage offered by the TruG is that the nonlinearity is learnable from data, alongside the model weights. Due to its stochastic nature, the TruG can be readily integrated into many stochastic neural networks for which hidden units are random variables. Extensive experiments have demonstrated significant performance gains that the TruG framework can bring about when it is integrated with the RBM, temporal RBM, or TGGM.

Acknowledgements The research reported here was supported by the DOE, NGA, NSF, ONR and by Accenture. 8

2 References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097?1105, 2012. [2] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251? 257, 1991. [3] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), pages 807?814, 2010. [4] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. Neural computation, 14(8):1771?1800, 2002. [5] Qinliang Su, Xuejun Liao, Chunyuan Li, Zhe Gan, and Lawrence Carin. Unsupervised learning with truncated gaussian graphical models. In The Thirty-First National Conference on Artificial Intelligence (AAAI), 2016. [6] Forest Agostinelli, Matthew D. Hoffman, Peter J. Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. CoRR, 2014. [7] Carson Eisenach, Han Liu, and ZhaoranWang. Nonparametrically learning activation functions in deep neural nets. In Under review as a conference paper at ICLR, 2017. [8] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In International Conference on Machine Learning (ICML), 2013. [9] Caglar Gulcehre, Kyunghyun Cho, Razvan Pascanu, and Yoshua Bengio. Learned-norm pooling for deep feedforward and

recurrent neural networks. In Machine Learning and Knowledge Discovery in Databases, pages 530?546, 2014. [10] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. Cognitive science, 9(1):147?169, 1985. [11] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527?1554, 2006. [12] Radford M Neal. Connectionist learning of belief networks. Artificial intelligence, 56(1):71?113, 1992. [13] Brendan J Frey. Continuous sigmoidal belief networks trained using slice sampling. In Advances in Neural Information Processing Systems, pages 452?458, 1997. [14] Qinliang Su, Xuejun Liao, Changyou Chen, and Lawrence Carin. Nonlinear statistical learning with truncated gaussian graphical models. In Proceedings of the 33st International Conference on Machine Learning (ICML-16), 2016. [15] Ilya Sutskever, Geoffrey E Hinton, and Graham W. Taylor. The recurrent temporal restricted boltzmann machine. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, Advances in Neural Information Processing Systems 21, pages 1601?1608. Curran Associates, Inc., 2009. [16] Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. Continuous univariate distributions, vol. 1-2, 1994. [17] Nicolas Chopin. Fast simulation of truncated gaussian distributions. Statistics and Computing, 21(2):275? 288, 2011. [18] Christian P Robert. Simulation of truncated normal variables. Statistics and computing, 5(2):121?125, 1995. [19] Ilya Sutskever and Geoffrey E Hinton. Learning multilevel distributed representations for high-dimensional sequences. In AISTATS, volume 2, pages 548?555, 2007. [20] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In Proceedings of the 25th international conference on Machine learning, pages 872?879. ACM, 2008. [21] David E Carlson, Edo Collins, Ya-Ping Hsieh, Lawrence Carin, and Volkan Cevher. Preconditioned spectral descent for deep learning. In Advances in Neural Information Processing Systems, pages 2971?2979, 2015. [22] Benjamin M Marlin, Kevin Swersky, Bo Chen, and Nando D Freitas. Inductive principles for restricted boltzmann machine learning. In International conference on artificial intelligence and statistics, pages 509?516, 2010. [23] Radford M Neal. Annealed importance sampling. Statistics and Computing, 11(2):125?139, 2001. [24] Roni Mittelman, Benjamin Kuipers, Silvio Savarese, and Honglak Lee. Structured recurrent temporal restricted boltzmann machines. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), pages 1647?1655, 2014. [25] Zhe Gan, Chunyuan Li, Ricardo Henao, David E Carlson, and Lawrence Carin. Deep temporal sigmoid belief networks for sequence modeling. In Advances in Neural Information Processing Systems, pages 2467?2475, 2015. [26] Jos? Miguel Hern?ndez-Lobato and Ryan P Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. Proceedings of The 32nd International Conference on Machine Learning, 2015. [27] Siamak Ravanbakhsh, Barnab?s P?czos, Jeff Schneider, Dale Schuurmans, and Russell Greiner. Stochastic neural networks with monotonic activation functions. AISTATS, 1050:14, 2016. [28] Max Welling, Michal Rosen-Zvi, and Geoffrey E Hinton. Exponential family harmoniums with an application to information retrieval. In NIPS, pages 1481?1488, 2004.

9

[29] Qinliang Su and Yik-Chung Wu. On convergence conditions of gaussian belief propagation. IEEE Transactions on Signal Processing, 63(5):1144?1155, 2015. [30] Qinliang Su and Yik-Chung Wu. Convergence analysis of the variance in gaussian belief propagation. IEEE Transactions on Signal Processing, 62(19):5119?5131, 2014. [31] Brendan J Frey and Geoffrey E Hinton. Variational learning in nonlinear gaussian belief networks. Neural Computation, 11(1):193?213, 1999. [32] Qinliang Su and Yik-Chung Wu. Distributed estimation of variance in gaussian graphical model via belief propagation: Accuracy analysis and improvement. IEEE Transactions on Signal Processing, 63(23):6258?6271, 2015. [33] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In Advances in Neural Information Processing Systems 27, pages 963?971. Curran Associates, Inc., 2014. [34] Soumya Ghosh, Francesco Maria Delle Fave, and Jonathan Yedidia. Assumed density filtering methods for learning bayesian neural networks. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI?16, pages 1589?1595, 2016.

10