Activation Adaptation in Neural Networks

Farnoush Farhadi[1], Vahid Partovi Nia[1,2][a] and Andrea Lodi[1][b]
[1] *Polytechnique Montreal,2900 Edouard Montpetit Blvd, Montreal, Quebec H3T 1J4, Canada*
[2] *Huawei Noah's Ark Lab, Montreal Research Centre, 7101 Park Avenue, Quebec H3N 1X9, Canada*

Keywords: Activation Function, Convolution, Neural Networks.

Abstract: Many neural network architectures rely on the choice of the activation function for each hidden layer. Given the activation function, the neural network is trained over the bias and the weight parameters. The bias catches the center of the activation, and the weights capture the scale. Here we propose to train the network over a shape parameter as well. This view allows each neuron to tune its own activation function and adapt the neuron curvature towards a better prediction. This modification only adds one further equation to the back-propagation for each neuron. Re-formalizing activation functions as a comulative distribution function (cdf) generalizes the class of activation function extensively. We propose to generalizing towards extensive class of activation functions and study: i) skewness and ii) smoothness of activation functions. Here we introduce adaptive Gumbel activation function as a bridge between assymmetric Gumbel and symmetric sigmoid. A similar approach is used to invent a smooth version of ReLU. Our comparison with common activation functions suggests different data representation especially in early neural network layers. This adaptation also provides prediction improvement.

1 INTRODUCTION

Neural networks achieved considerable success in image, speech, and text classification. In many neural networks only bias and weight parameters are learned to fit the data, while the activation function of each neuron is pre-specified to sigmoid, hyperbolic tangent, ReLU, etc. From a theoretical standpoint, a neural network reasonably wide and deep, approximates an arbitrarily complex function independent of the chosen activation function, see (Hornik et al., 1989) and (Cho and Saul, 2010). However, in practice, the prediction performance and the learned representation depends on hyperparameters such as network architecture, number of layers, regularization function, batch size, initialization, activation function, etc. Despite large studies on network hyperparameter tuning, there have been few studies on how to choose an appropriate activation function. The choice of activation function changes learning representation and also affects the network performance (Agostinelli et al., 2014). We propose let data estimate the activation function during training by developing a flexible activation function. We demonstrate how to formalize this such activations and show how to embed it in the back-propagation.

Developing an adaptive activation function helps fast training of deep neural networks, and has attracted attention, see for instance (Zhang and Woodland, 2015), (Agostinelli et al., 2014), (Agostinelli et al., 2014), (Jarrett et al., 2009), (Glorot et al., 2011), (Goodfellow et al., 2013), (Springenberg and Riedmiller, 2013) and recently (Dushkoff and Ptucha, 2016), (Hou et al., 2016), (Hendrycks and Gimpel, 2016), (Hou et al., 2017) and (Klambauer et al., 2017). Here, we introduce adaptive activation functions by combining two main tools: i) looking at activation as a cumulative distribution function (cdf) and ii) making an adaptive version by equipping a distribution with a shape parameter. The shape parameter is continuous, so that an update equation can be added in back-propagation. Here, we focus on the simple architecture of LeNet5, but this idea can be used to equip more complex and deep architectures with flexible activations (Ramachandran et al., 2018).

There has been a surge of work in modifying the ReLU. Leaky ReLU is one of the most famous modifications that gives a slight negative slope on a negative argument (Maas et al., 2013). Another modification called ELU (Clevert et al., 2015) attempts exponen-

[a] https://orcid.org/0000-0001-6673-4224
[b] https://orcid.org/0000-0002-5101-095X

tial decrease of the slope from a predefined value to zero. Taking a mixture approach, (Qian et al., 2018) proposed a mixed function of leaky ReLU (Maas et al., 2013) and ELU (Clevert et al., 2015) as an adaptive function, that could be learned in a data-driven way unlike SeLU (Klambauer et al., 2017). Inspired by (Agostinelli et al., 2014), (Zhang and Woodland, 2015) (Qian et al., 2018) and (Klambauer et al., 2017), we study the effect of i) the asymmetry and ii) the smoothness of activation function. Our approach can build recntly proposed modifications for quantized training such as SignSWISH of (Darabi et al., 2019) and foothill of (Belbahri et al., 2019). The first study is performed by introducing an adaptive asymmetric Gumbel activation that changes its shape towards the symmetric sigmoid function. The second study is achieved by equipping the ReLU function with a smoothness parameter which generalizes the existing adaptive activations through distribution functions. In both cases, we tune the shape parameter for each neuron independently by adding an updating equation to back-propagation.

The performance of two fully-connected neural networks and a convolutional network are compared on simulated data, MNIST benchmark, and Movie review sentiment data. As an application, we use the classical LeNet5 architecture (Kim, 2014) to classify the users' intention using URLs they navigated on their browser.

2 ADAPTIVE ACTIVATIONS

We recommend to perceive the activation function as a cumulative distribution function bounded on $[0,1]$. Common activation functions are bounded, but not necessarily to $[0,1]$ like hyperbolic tangent. A location and scale transformation is sufficient to transform their range if necessary. However, still widely-used activations such as ReLU or leaky ReLU are unbounded. One may decompose unbounded activation functions into two components: i) a bounded component and ii) an unbounded component, and only adapt the bounded ingredient through a continuous cumulative distribution function.

2.1 Adaptive Gumbel

One of the common activation functions is the sigmoid function that maps a real value to $[0,1]$ similar to cumulative distribution functions. Although sigmoid is rarely used in convolution layers, still it is widely used in softmax layer and attention type models that revolutionized in natural language processing (Vaswani et al., 2017). More precisely, the sigmoid function is the cumulative distribution function of the symmetric and bell-shaped logistic distribution. An asymmetric activation can be developed using a cumulative distribution function of a skewed distribution like Gumbel, for example. Gumbel distribution is the asymptotic distribution of extreme values such as minimum or maximum (Coles et al., 2001). A shape parameter that pushes a sigmoid distribution function away from the logistic and more towards a Gumbel distribution defines a sort of a shape parameter. Our proposed adaptive Gumbel activation is

$$\sigma_\alpha(x) = 1 - \{1 + \alpha \exp(x)\}^{-\frac{1}{\alpha}} \quad \alpha \in \mathbb{R}^+, x \in \mathbb{R}. \tag{1}$$

The above form is inspired by Box-Cox transformation (Box and Cox, 1964) for binary regression. The simplest form of neural network with no hidden layer is a binary regression in which (1) generalizes logistic regression towards complementary log-log regression by tuning $\alpha \in (0,1]$, see Figure 1 (left panel). The Gumbel cumulative distribution function arises in the limit while $\alpha \to 0$. The foothill function of (Belbahri et al., 2019) and SignSwish function of (Darabi et al., 2019) can be re-formalized in this context.

2.2 Adaptive ReLU

The ReLU activation function

$$\sigma(x) = \max(0, x)$$

is unbounded, unlike the sigmoid or hyperbolic tangent. One may re-write the ReLU activation as

$$\sigma(x) = x\Delta(x),$$

where $\Delta(x)$ is the cumulative distribution function of a degenerate distribution. The function $\Delta(x)$ is also known as *Heaviside* function and coincides with the integral of the *Dirac delta* function. We propose to replace $\Delta(x)$ with a smooth cumulative distribution function $\Delta_\alpha(x)$ such as the exponential cumulative distribution function

$$\begin{aligned} \Delta_\alpha(x) &= (1 - e^{-\alpha x})\mathbb{I}_{\{x>0\}}(x), \\ \sigma_\alpha(x) &= x\Delta_\alpha(x), \quad \alpha \in \mathbb{R}^+, x \in \mathbb{R}, \end{aligned} \tag{2}$$

where $\mathbb{I}_A(x)$ is the indicator function on set A.

In (2) we recommend to equip the degenerate distribution $\Delta(x)$ with a smoothing parameter α. Any continuous random variable with a scale parameter is a convenient choice for $\Delta_\alpha(x)$. A random variable with infinitesimal scale behaves like a degenerate distribution, so $\Delta(x)$ is retrieved when the scale tends to zero, or equivalently $\alpha \to \infty$. The generalized ReLU (2) coincides with the SWISH activation

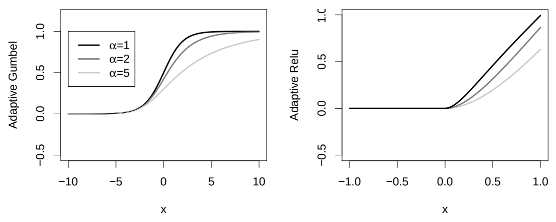

Figure 1: Adaptive Gumbel activation (left panel) and adaptive ReLu activation (right panel) for $\alpha = 1, 2, 5$.

function (Ramachandran et al., 2018) if $\Delta_\alpha(x)$ is the logistic cumulative distribution function

$$\Delta_\alpha(x) = (1 + e^{-\alpha x})^{-1}. \tag{3}$$

The SiLU (Elfwing et al., 2018) is a special case of (3) while $\alpha = 1$.

One may show that the proposed parameterization preserves model identifiability in simple Bernoulli regression model.

Theorem 1. *a binary regression with the adaptive activation* (1) *is identifiable.*

See Appendix for the proof.

3 BACK-PROPAGATION

Define the vector of linear predictors of layer l as

$$\eta^l = [\eta_1^l, \ldots, \eta_n^l]^\top,$$

where

$$\eta_i = w_0 + \mathbf{w}^\top \mathbf{x}_i, i = 1, \ldots, n,$$

and the lth hidden layer output $\mathbf{h}^l = \sigma(\eta^l)$, in which $\eta^l = w_0 + \mathbf{w}^\top \mathbf{h}^{l-1}$.

Traditionally, $\sigma(x)$ is sigmoid or ReLU activation. The adapted back-propagation uses the conventional back-propagation, but each neuron carries its own activation function $\sigma_\alpha(x)$. The adaptation parameter α for each neuron is trained along with bias and weights $[w_0, \mathbf{w}]^\top$.

Suppose the vector of parameters for a neuron in layer l is $\theta^l = [\alpha^l, w_0^l, \mathbf{w}^l]$ and the network is trained using loss function $\mathcal{L}(.)$.

In practice, $\mathcal{L}(.)$ is the entropy loss for classification, and the squared error loss for regression, penalized with an L_2 norm $\sum_j \theta_j^2$ or an L_1 norm $\sum_j |\theta_j|$ upon convenience. The updating back-propagation rule, given a learning rate $\gamma > 0$, for a neuron in layer l is

$$w_0^l \leftarrow w_0^l - \gamma \frac{\partial \mathcal{L}}{\partial w_0^l}, \tag{4}$$

$$\mathbf{w}^l \leftarrow \mathbf{w}^l - \gamma \frac{\partial \mathcal{L}}{\partial \mathbf{w}^l}, \tag{5}$$

$$\alpha^l \leftarrow \alpha^l - \gamma \frac{\partial \mathcal{L}}{\partial \alpha^l}, \tag{6}$$

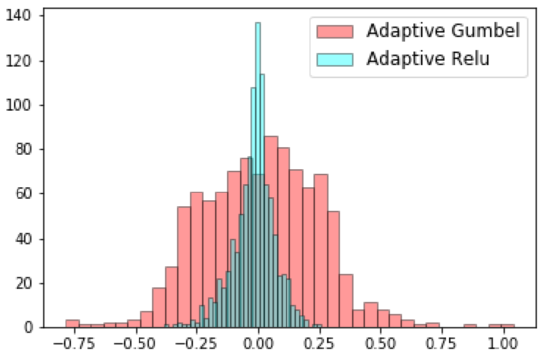

Figure 2: Histogram of fitted α for Adaptive ReLu (blue) and Adaptive Gumbel (red) activation functions for simulated data with one hidden layer.

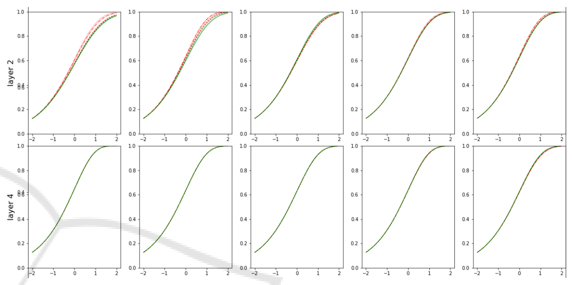

Figure 3: Fitted adaptive Gumbel activations on layer 2 and layer 4 of an eight hidden layer network. Adapted activations vary more often in earlier layers, see also Figure 4.

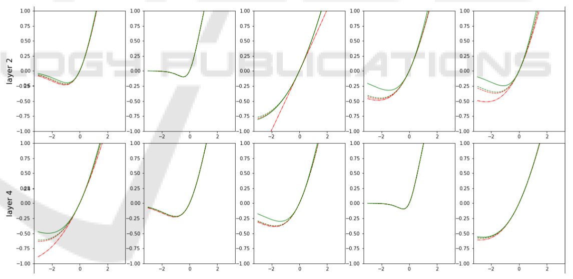

Figure 4: Fitted adaptive ReLU activations on layer 2 and layer 4 of an eight hidden layer network. Adapted activations vary more often in earlier layers, see also Figure 3.

where (4) updates the bias, (5) updates the weights, and (6) adapts the activation function. Note that one may choose different learning rates for each equation. We recommend to reparametrize (6) with e^α in numerical computations to enforce $\alpha > 0$.

4 BENCHMARKS

Here, we compare the adaptive modification in three datasets. One dataset is a simulated fully-connected network in Section 4.1, where the true activation function and true labels are known. Furthermore, we evaluate activation adaptation on convolutional models on the MNIST image data in Section 4.2, and on Movie Review text data in Section 4.3.

4.1 Simulated Data

Our objective in this experiment is to understand how the choice of activation function affects the performance of the network. We simulated data from two fully-connected neural networks: i) a network with only 1 hidden layer, and ii) with 8 hidden layers, each layer with 10 neurons. Activation functions are fixed to ReLU and sigmoid in data simulation setup. According to works in (LeCun et al., 1998b) and recently in (Krizhevsky et al., 2012) and (Glorot and Bengio, 2010), several weight initialization and different combination of number of input and output units in weight initialization formulation along with different type of activation functions could be employed in deep neural networks. In this paper, biases were initialized from $\mathcal{N}(0,0.5)$ in simulated models and biases in fitted models are initialized by formulation suggested in (LeCun et al., 1998b). The original weights in simulated models were initialized from a mixture of two normals $\mathcal{N}_1(1,0.5)$ and $\mathcal{N}_2(-1,0.5)$ with an equal proportion and weights in fitted models initialized by (LeCun et al., 1998b) settings.

The simulated data set includes $10'000$ examples of 10 features with a binary output. In each configuration, we have a fully-connected network with 10 neurons at each layer is trained with a fixed learning rate $\gamma = .01$, regularization parameters $L_1 = .001$ and $L_2 = .001$, batch size = 20, and number of epochs = 2000 for both simulated and fitted models. The learning rate, L_1 and L_2 regularization constants are tuned using 5-fold cross-validation. The average results are reported from a 5-fold cross validation as well.

The results summarized in Table 1 show that, overall, adaptive Gumbel outperforms sigmoid. Adaptive ReLU competes closely with ReLU in shallow networks, and slightly outperforms ReLU in deeper networks. Figure 2 confirms adaptive Gumbel and adaptive ReLU have different training range for α. Figure 3 and Figure 4 depicts the learned activations in an eight hidden layer fully connected network. Early layers have more variable learned activations, and often the last layers do not change much. The deeper layers are more difficult to learn.

We run several models with different number of fully connected hidden layers from 1 to 8 including 1, 2, 4 and 8 layers. In Table 1, we aimed at studying the effect of varying activation functions on the accuracy of the predictions produced by the fitted model compared to its original counterpart when they have the same number of hidden layers, same number of neurons at each layer. Our goal was to understand how accurately fitted model could predict the class labels at the end of training step while we increase the number of hidden layers from 1 (as a simple fully connected multi-layer perceptron) to a very deeper one with 8 hidden layers. In this paper we only present the results for 1 and 8 hidden layers due to page limits. For models with less layers, fitting models with ReLU or adaptive ReLU almost outperforms the other activation functions. For deeper models, fitting models with the adaptive Gumbel also exhibits a good performance in competition with ReLU and adaptive ReLU. As seen in this table, as the number of hidden layers increases, an expected drop in the performance of all fitted models is observed. Hence, it is evident that there is no question to continue for deeper fully-connected layers.

Table 1: Prediction accuracy for data simulated with sigmoid (Sig), adaptive Gumbel (AGumb), ReLU and adaptive ReLU (AReLU) in networks with 1 and 8 hidden layers. The maximum standard error is 0.22.

simulated		fitted network			
layers		Sig	AGumb	ReLU	AReLU
1	Sig	95.8	97.5	**97.8**	97.7
	ReLU	96.1	97.4	**98.2**	98
8	Sig	83.7	**83.8**	81.8	81.9
	ReLU	57.3	88.2	89.3	**89.9**

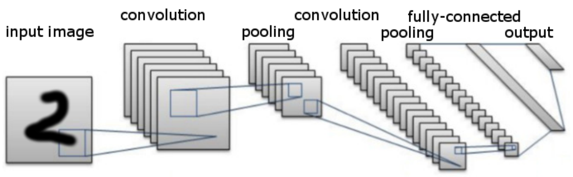

Figure 5: The LeNet5 architecture. Two convolutional layers, each layer followed by a max polling, and eventually a fully-connected layer on top.

4.2 MNIST Data

Here we evaluate the performance of adapting activation on convolutional neural networks using handwritten digits grayscale image data. Our convolutional architecture is the classical LeNet5 (LeCun et al., 1998a), but with adaptive activations. The original LeNet5 use a modified hyperbolic tangent activation, but the ReLU activation is often used which provides a superior classification accuracy. The LeNet5 architecture contains two convolutional layers, each convolutional layer followed by a max-pooling layer. A single fully-connected hidden layer is put on top

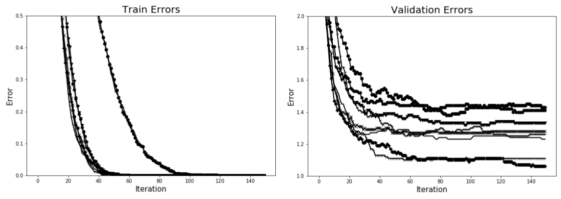

Figure 6: Training and validation error curves of different adaptive models on MNIST data.

with 1000 neurons, see Figure 5.

Motivated from Section 4.1, we only keep ReLU as the strong competitor, because adaptive Gumbel always outperforms sigmoid. The network parameters are trained with batch normalization, learning rate $\gamma = 0.01$, batch size 100, and iterated 150 epochs. Figure 8 (left panel) shows are adaptive models converge. Prediction accuracy is summarized in Table 2. The best performance appears for adaptive Gumbel on convolutional layer, closely followed by ReLU. Hyper parameters including learning rate for CNN models are chosen according to the primary settings of LeNet5 implementation developed by Theano development team. We select fixed hyper parameters in all CNN models to study the effect of changing activation functions on final performance. The accuracy of each model is reported by running the corresponding algorithm on standard test set provided in MNIST data.

Table 2: Convolutional architecture of LeNet5 on MNIST data. The activations in the rows represent the activation functions of convolutional layers, while the columns represent activations of fully-connected layers for ReLU, adaptive ReLU (AReLU), and adaptive Gumbel (AGumb) activations. The sigmoid activation is not reported as it was beaten always by the other techniques. The accuracy is computed over the standard test set.

Conv layer	fully-connect layer		
	ReLU	AReLU	AGumb
ReLU	99.1	98.7	**99.1**
AReLU	98.8	98.9	98.9
AGumb	98.7	98.8	98.9

4.3 Movie Review Data

This time we try convolutional architecture on text data over pre-trained word vectors. The data consists of 2000 movie reviews, 1000 positive and 1000 negative (Pang and Lee, 2004).

These word vectors use the word2vec (Mikolov et al., 2013) trained on 100 billion words of Google News to embed a word in a vector of dimension 300. Word2vec transforms each word into a vector such that the words semantics is preserved. A CNN model with pre-trained word2vec vectors called static in (Kim, 2014), is used in our experiments. In this variant, the static CNN we use involves two convolutional layers each of them followed by a max-pooling layer and a fully-connected layer at the end. The fully-connected network includes one hidden layer with 100 neurons and a softmax output layer for binary text classification. The hyper parameters in all models are same including learning rate $\gamma = 0.05$, image dimensions (img-w) = 300, filter sizes = $[3, 4, 5]$, each have 100 feature maps, batch size = 50, dropout = 0.5, number of hidden layers = 1, number of neuron = 100 and number of epochs = 50. For consistency, same data, pre-processing and hyper-parameter settings are used as reported in (Kim, 2014). Unlike MNIST data, the Movie Review dataset does not have a standard test set. So, we report the average accuracy over 5-fold cross-validation in Table 3.

Again adaptive activation provides a better prediction accuracy. Figure 8 (right panel) suggests that for Movie review data is more difficult to converge compared to MNIST. We suspect this happens, because i) text data carry less information compared to image, ii) embedding may mas some information that exists in the text.

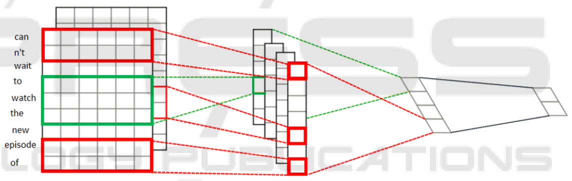

Figure 7: Movie comments are embedded into a vector. Then a CNN model classifies the text to a ”positive review” or ”negative review”.

Table 3: Convolutional architecture of LeNet5 on Movie review data. The activations in the rows represent the activation functions of convolutional layers, while the columns represent activations of fully-connected layers for ReLU, adaptive ReLU (AReLU), and adaptive (AGumb) Gumbel activations. The maximum standard error estimated using 5-fold cross-validation is 0.17.

Conv layer	Fully-connect layer		
	ReLU	AReLU	AGumb
ReLU	79.1	79.1	78.9
AReLU	78.9	78.5	**79.3**
AGumbel	52.3	77.4	78.7

Accordingly, our empirical results show that using adaptive Gumbel, as the activation function, in fully-connected layer is a good choice. Moreover, adaptive ReLU works when it is applied in convolutional layers. A comparison between the best results of our experiments (reported from 5-fold crossvalidation) and the state-of-the-art results by (Kim, 2014)

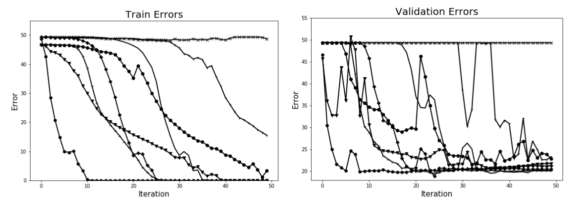

Figure 8: Training and validation error curves of different adaptive models on Movie review data.

on the same version of Movie Review data shows that our adaptive activation functions perform fairly good and are match on Movie Review data in terms of accuracy. Applying those adaptive activation functions with fine-tuning the hyper parameters may result in further improvement.

5 APPLICATION

The sequence of URL clicks are gathered to study user intentions by an international tech company. The dataset is private and is extracted from a survey, while anonymous visitors visited a commercial website over a period of three months and their action is recorded at the end of the session. In those surveys, visitors are typically asked to tell their purpose of visit from the "browse-search product", "complete transaction-purchase", "get order-technical support" or "other", so treated as a four-class problem. The study aims at exploring the relationship between user behavioral data (which includes URL sequences as features). The stated purpose of the visit provided in the survey after the end of sessions. The data consist of approximately 13500 user sessions and each record is limited to maximum of 50 page visits. The goal is to construct a model that predicts visitors' intention based on URL sequence they navigated from page to page.

The application on user intention prediction are just about to make their early steps, see (Liu et al., 2015), (Vieira, 2015), (Korpusik et al., 2016), (Lo et al., 2016), and (Hashemi et al., 2016). Our approach in this work is to use neural networks in two steps; like in the Movie review data of Section 4.3, by i) embedding a URL into representative vectors and ii) using these representations as features to a neural network to predict the user intention. We treat each URL as a sentence, where each word of this sentence is separated by "/". A similar approach is used in text classification, sentiment analysis (Kim, 2014), semantic parsing (Yih et al., 2014) and sentence modeling, see (Kalchbrenner et al., 2014) and (Kim, 2014). Again we applied LeNet5 architecture with adaptive activation. We report *precision* and *recall*, because different methods compete very closely in terms of accuracy.

The results summarized in Table 4 show that adaptive Gumbel performs slightly better, while ReLU and adaptive ReLU closely compete with each other.

Table 4: Running LeNet5 convolutional model with ReLU, adaptive ReLU (AReLU), and adaptive Gumbel (AGumb) activations on URL data. Different activations performed almost identical in terms of accuracy, so only the prescition (P), and the recall (R) are reported.

Conv layer	Fully-connect layer					
	ReLU		AReLU		AGumb	
	P	R	P	R	P	R
ReLU	68.0	68.0	68.1	67.9	68.0	67.8
AReLU	68.1	67.9	68.0	68.0	67.9	67.9
AGumb	68.1	68.3	68.2	68.1	68.1	**68.3**

5.1 Conclusion

We proposed a general method to adapt activation functions by looking at the activation function as a cumulative distribution function. This view is useful to adapt a bounded activation such as sigmoid or hyperbolic tangent commonly used in recurrent neural networks and attention models. It is well-known in deep neural networks with bounded activations suffer from *vanishing gradient*. Therefore, a methodology for adapting unbounded activations is as important. We recommended to decompose ReLU into an unbounded component and a bounded component. Therefore, the cumulative distribution function idea can be re-used to adapt the bounded counterpart.

In fully-connected networks adapting activation helps prediction most of the time. According to our experiments adapting activation helps prediction accuracy often, even in complex architectures. However, adaptive Gumbel mostly outperforms other activations in convolutional architectures.

We designed a series of experiments to understand how our proposed activation functions affect on the prediction of the fitted models using a simulated data. The results of this experiment show that *using adaptive activation functions in fitting models for prediction approximation is superior compared to standard functions in terms of accuracy regardless of network size and activation functions used in original model.* In the next step, we aim at evaluating the performance of the typical convolutional neural network (CNN) models using our proposed activation functions on image and text data. Accordingly, we design a series of experiments on MNIST as a widely-used image benchmark to understand how accurate the adaptive activation functions in LeNet5 CNN models classify the hand-written digit images compared to standard activation functions. Compared to standard sigmoid, applying adaptive Gumbel in fully-connected layer of the CNN models are recommended. Generally, CNN models using proposed activation functions improves prediction and convergence speed compared to models that work exclusively with standard functions.

A series of experiments using CNN models trained on a top of word2vec text data is performed to evaluate the performance of the proposed activation functions in a sentiment analysis application on Movie Review benchmark. Our empirical results imply that using adaptive Gumbel as activation functions in fully-connected layer and adaptive ReLU in convolutional layers are strongly recommended. These observations are consistent with the findings were noticed in experiments on MNIST data. Also, a comparison between our best observations and the state-of-the-art results in (Kim, 2014) indicates that our reported accuracy using adaptive activation functions reproduces their accuracy. We believe that applying more fine-tuning hyper parameters and using other complex variants of CNN models accompanied with our proposed activation functions could improve the existing results. To recap, our experiments on two well-known image and text benchmarks imply that by virtue of using adaptive-activation functions in CNN models, we can improve the performance of the deep networks in terms of accuracy and convergence.

Learning the adaptation parameter is feasible by adding only one equation to back-propagation. Computationally, letting neurons of a layer choose their own activation function, in this framework, is equivalent to adding a neuron to a layer. This minor extra computation changes the network flexibility considerably, especially in shallow architectures. We focused only on the classic LeNet5 architecture, but there is a potential of exploring this methodology with a wide variety of distribution functions for portable architectures such as MobileNets (Howard et al., 2017), ProjectionNets (Ravi, 2017), SqueezeNets (Iandola et al., 2016), QuickNets (Ghosh, 2017), etc. It also has a potential to generalize modified quantized training (Hubara et al., 2018) and (Partovi Nia and Belbahri, 2018).

REFERENCES

Agostinelli, F., Hoffman, M., Sadowski, P., and Baldi, P. (2014). Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Belbahri, M., Sari, E., Darabi, S., and Partovi Nia, V. (2019). Foothill: A quasiconvex regularization for edge computing of deep neural networks. In *International Conference on Image Analysis and Recognition*, pages 3–14.

Box, G. E. and Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252.

Cho, Y. and Saul, L. K. (2010). Large-margin classification in infinite neural networks. *Neural Computation*, 22(10):2678–2697.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Coles, S., Bawa, J., Trenner, L., and Dorazio, P. (2001). *An introduction to statistical modeling of extreme values*. Springer.

Darabi, S., Belbahri, M., Courbariaux, M., and Partovi Nia, V. (2019). Regularized binary network training. In *Neural Information Processing Systems*, Workshop on Energy Efficient Machine Learning and Cognitive Computing.

Dushkoff, M. and Ptucha, R. (2016). Adaptive activation functions for deep networks. *Electronic Imaging*, 2016(19):1–5.

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*.

Ghosh, T. (2017). Quicknet: Maximizing efficiency and efficacy in deep architectures. *arXiv preprint arXiv:1701.02291*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. *arXiv preprint arXiv:1302.4389*.

Hashemi, H. B., Asiaee, A., and Kraft, R. (2016). Query intent detection using convolutional neural networks. In *International Conference on Web Search and Data Mining, Workshop on Query Understanding*.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hou, L., Samaras, D., Kurc, T., Gao, Y., and Saltz, J. (2017). Convnets with smooth adaptive activation functions for regression. In *Artificial Intelligence and Statistics*, pages 430–439.

Hou, L., Samaras, D., Kurc, T. M., Gao, Y., and Saltz, J. H. (2016). Neural networks with smooth adaptive activation functions for regression. *arXiv preprint arXiv:1608.06557*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam,

H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G.-H. (2005). Model identifiability. *Wiley StatsRef: Statistics Reference Online*.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2018). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(187):1–30.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980.

Korpusik, M., Sakaki, S., Chen, F., and Chen, Y.-Y. (2016). Recurrent neural networks for customer purchase prediction on twitter. In *CBRecSys@ RecSys*, pages 47–50.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998b). Efficient backprop. In *Neural networks: Tricks of the Trade*, pages 9–50. Springer.

Liu, Q., Yu, F., Wu, S., and Wang, L. (2015). A convolutional click prediction model. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1743–1746. ACM.

Lo, C., Frankowski, D., and Leskovec, J. (2016). Understanding behaviors that lead to purchasing: A case study of pinterest. In *KDD*, pages 531–540.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics.

Partovi Nia, V. and Belbahri, M. (2018). Binary quantizer. *Journal of Computational Vision and Imaging Systems*, 4(1):3–3.

Qian, S., Liu, H., Liu, C., Wu, S., and San Wong, H. (2018). Adaptive activation functions in convolutional neural networks. *Neurocomputing*, 272:204–212.

Ramachandran, P., Zoph, B., and Le, Q. V. (2018). Searching for activation functions.

Ravi, S. (2017). Projectionnet: Learning efficient on-device deep networks using neural projections. *arXiv preprint arXiv:1708.00630*.

Springenberg, J. T. and Riedmiller, M. (2013). Improving deep neural networks with probabilistic maxout units. *arXiv preprint arXiv:1312.6116*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Vieira, A. (2015). Predicting online user behaviour using deep learning algorithms. *arXiv preprint arXiv:1511.06247*.

Yih, S. W.-t., He, X., and Meek, C. (2014). Semantic parsing for single-relation question answering.

Zhang, C. and Woodland, P. C. (2015). Parameterised sigmoid and relu hidden activation functions for dnn acoustic modelling. In *Sixteenth Annual Conference of the International Speech Communication Association*.

APPENDIX

Proof of Theorem 1. Suppose the Bernoulli distribution

$$p(y_i) = \pi_i^{y_i}(1-\pi_i)^{(1-y_i)} \quad , \quad y_i = \{0,1\},$$

where π_i is a function of parameters $\omega = (\eta, \alpha)$ where η is the linear predictor, and α is the activation shape

$$\pi_i = \sigma_\alpha(\eta) \tag{7}$$

Therefore, to ensure distinct probability distributions are indexed by α on a continuum of $\alpha > 0$, the identifiability of $\sigma_\alpha(x)$ must be studied, i.e. distinct values of parameter α lead to distinct activation functions. Formally,

$$\alpha \neq \alpha' \leftrightarrow \exists x \in \mathbb{R} \text{ s.t. } \sigma_\alpha(x) \neq \sigma_{\alpha'}(x). \tag{8}$$

Equivalently

$$\sigma_\alpha(x) = \sigma_{\alpha'}(x) \leftrightarrow \alpha = \alpha' \tag{9}$$

which falls on $\sigma_\alpha(x)$ identifiability definition (Huang, 2005). Take $\sigma_\alpha(x)$ in equation (1) that defines adaptive Gumbel

$$\begin{aligned} \sigma_\alpha(x) &= \sigma_{\alpha'}(x), \\ \{1+\alpha\exp(x)\}^{\frac{1}{\alpha}} &= \{1+\alpha'\exp(x)\}^{\frac{1}{\alpha'}}. \end{aligned}$$

Given $1+\alpha\exp(x)>0$

$$\frac{1}{\alpha}\log\{1+\alpha\exp(x)\} = \frac{1}{\alpha'}\log\{1+\alpha'\exp(x)\}.$$

Let $z=\exp(x)$ and $f_z(\alpha)=\log(1+\alpha z)$ which has derivative of arbitrary order. Take the Taylor expansion of $\log(1+\alpha z)$

$$\frac{1}{\alpha}\left\{\sum_{n=0}^{\infty}\frac{f_z^{(n)}(0)\alpha^n}{n!}\right\} = \frac{1}{\alpha'}\left\{\sum_{n=0}^{\infty}\frac{f_z^{(n)}(0)\alpha'^n}{n!}\right\}$$

The latter equation holds for all z if and only if all corresponding polynomial coefficients are equal. Equivalently,

$$\alpha=\alpha'. \tag{10}$$