International Journal on Artificial Intelligence Tools
Vol. 30, No. 2 (2021) 2150008 (13 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218213021500081

World Scientific
www.worldscientific.com

Hyperparameter-free Regularization by Sampling from an Infinite Space of Neural Networks

Thomas M. Whitehead
Intellegens Ltd., Eagle Labs, Chesterton Road, Cambridge, CB4 3AZ, United Kingdom
tom@intellegens.ai

Received 2 October 2020
Accepted 15 December 2020
Published 26 March 2021

Stochastic activation functions, where the output of an activation function is a hyperparameter-free random function of the inputs, generalize the concept of dropout to sampling from an infinitely large space of related networks. Stochastic activation functions provide intrinsic regularization and sparsification of artificial neural networks, along with cheap and accurate estimates of the uncertainty in the predictions from a network. Examples are presented against standard benchmarking datasets.

Keywords: Artificial neural networks; activation functions; regularization; dropout; uncertainty quantification.

1. Introduction

In recent years deep neural networks have revolutionized the field of machine learning, enabling computers to compete with, and in many cases exceed, humans at tasks as diverse as speech recognition,[1] semantic image recognition,[2] and the playing of complex strategy games.[3] The depth, and success, of these artificial neural networks comes from the layering of multiple steps of data abstraction and processing together into one adaptable model. When even one (sufficiently large) layer is capable of representing any functional form,[4] these multi-layer networks have the potential to flexibly extract and process compound structures from their training data.

However, this very flexibility can prove a hindrance in the pursuit of a generalizable model that is capable of extrapolating beyond its training data. The problem of a neural network "overfitting" to its training data, capturing the random noise as well as signal, has had many proposed solutions over the years, ranging from the penalizing of overly complex models,[5] through efficiently sampling from a large class of possible models,[6] to the augmentation of the training data itself.[7] In this paper we discuss a generalization of the "dropout" method of Ref. 6 that involves sampling from an infinitely, rather than exponentially, large space of connected neural network architectures.

Non-linear activation functions are at the core of the flexibility of neural networks. These functions combine features of the input data, or earlier layers of abstraction, to create representations of the data that can be efficiently extracted and classified. Activation functions in a hidden layer generally take the form $f(W_{ij}x_j + b_i)$ with inputs x_j, where summation is implicit over the repeated index j and the activation function f is applied elementwise to the resulting vector. W_{ij} and b_i, the weights and biases of the hidden layer's nodes, are optimizable parameters, which if f is differentiable are generally optimized by same variant of backpropagation.[8]

Common classes of activation functions include sigmoid functions, such as the hyperbolic tangent or softsign, and rectifiers, such as the rectified linear unit (ReLU) and scaled exponential linear unit.[9] The ReLU is a particularly popular activation function[10] due to its simplicity, with functional form $f(z) = \max(0, z)$, simple derivative $f'(z) = \Theta(z)$, where the Heaviside function $\Theta(z) = 1$ for $z > 0$ and $\Theta(z) = 0$ for $z \leq 0$, and lack of saturation: for sigmoid activation functions $f'(z)$ becomes small far from the origin, decelerating optimization using gradient-based methods, whilst the ReLU has constant gradient almost everywhere.

In this paper we describe a stochastic activation function that implements a sampling from an infinitely large space of neural network architectures. The motivation for this activation function is discussed in Sec. 2, with experimental results on a variety of standard benchmarking datasets in Sec. 3,[a] and conclusions in Sec. 4.

2. Stochastic Activation Function

The activation function of interest is a generalization of a standard activation function, taking the form

$$s_f(z) \equiv f(z) \times \mathcal{S}, \tag{1}$$

where $f(z)$ is any standard activation function and $\mathcal{S}$ is a random variable with expected value of unity. The output of the stochastic activation function $s_f(z)$ is then a random variable with the same expected activation as the underlying function $f(z)$, but with a range dependent on the form of $\mathcal{S}$. In all examples in this paper $f(z)$ takes a linear or rectified linear form (explicitly, $f(z) = z$ or $f(z) = \max(0, z)$). Treating this as the cumulative distribution function for the output value of the activation function, a natural choice is the corresponding probability density function, $\mathcal{S} \sim \mathcal{U}(0, 2)$, where $\mathcal{U}(0, 2)$ is the uniform random distribution between 0 and 2 (which has expected value of unity, as required; the lower limit of 0 corresponds to maintaining the sign of the underlying activation function). This choice corresponds to the maximum entropy choice for a multiplicative term

[a]Code freely available at https://github.com/tangoed2whiskey/StochasticActivationFunction.

that retains the expected value and sign of the underlying activation function. The explicit form of the stochastic activation function used in this work is then

$$s_f(z) \equiv f(z) \times \mathcal{U}(0,2). \tag{2}$$

When making new predictions the random variable $\mathcal{S}$ is generally replaced by its mean of unity, although retaining it provides access to a measure of uncertainty quantification (see Sec. 3.3).

The motivation for this activation function is three-fold. First, the injection of noise into neural networks has long been known to act as a regularizer that mitigates overfitting.[11] However, the noise scale is generally set as an external hyperparameter, independently of the value of the activation function or its argument, with no *a priori* understanding of the noise scale. This increases the cost of the training procedure, as the noise scale needs to be empirically determined along with the other hyperparameters of the system. With the stochastic activation function, on the other hand, the scale of the random variation is set by the scale of the inputs to the activation function. As the underlying activation function has no intrinsic length scale for linear or ReLU activations, this parameter-free noise scale fits well with the underlying activation function.

Secondly, the random rescaling of the activation function is equivalent to a random sampling of the infinitely large set of neural networks that share a common architecture with weights and biases connected by multiplicative transformations. This provides an implicit averaging over an infinite ensemble of models, which can reduce the variance of predictions without impacting the bias.[13] In contrast to standard ensemble-averaging techniques, however, stochastic activation functions require the training of only one model, rather than training each element of the ensemble separately.

This is a generalization of the well-established concept of dropout, where the output of an activation function is multiplied by a normalized Bernoulli random variable $\mathcal{S} \sim \mathcal{B}(p)$, giving

$$s_f^{\text{dropout}}(z) = f(z) \times \mathcal{B}(p), \tag{3}$$

where $\mathcal{B}(p)$ takes the value 0 with probability $1-p$ and value $1/p$ with probability p, such that the mean activation is preserved.[6] The hyperparameter p, the expected fraction of activities that are retained through dropout, must again be set empirically, and does not depend on the scale of the input or output of the activation function itself. Dropout efficiently approximates sampling from the exponentially large set of network architectures that share subsets of nodes, with weights shared between the networks. This can be viewed as an approximation to a full Bayesian neural network,[6,12] with much reduced computational expense.

The chief differentiators in using a uniform random variable instead of a Bernoulli random variable (or Gaussian random variable, as also seen in Ref. 6) are the lack of hyperparameters in the uniform random variable, which removes a degree of freedom from hyperparameter optimization for increased efficiency, and the direct interpretability in terms of sampling from an infinite space of network architectures. Gaussian dropout also samples from an infinite space of connected network architectures, but using a less intuitive biased sampling, which hinders interpretability.

The third motivation builds upon the idea of model averaging, as the network sampling implicit in the stochastic activation functions provides immediate access to a measure of the uncertainty in the predictions of the artificial neural network through the variance between the results of the networks in the ensemble. The quantification of uncertainty and confidence in predictions is vital for the application of artificial neural networks in many areas of science and technology.[14–16] Stochastic activation functions provide access to estimates of the confidence in predictions without any extra computational expense in training.

Experimental support is given to each of these motivations in the following Section.

3. Experimental Results

In this Section we provide experimental evidence to support the effectiveness of the stochastic activation function, in comparisons with standard methods against publically-available data sets. In Sec. 3.1 we examine how the stochastic activation function self-regularizes without the need for hyperparameter optimization; in Sec. 3.2 we identify how the stochastic activation function provides many of the same benefits as dropout, in terms of regularization but also sparsification of weights; and in Sec. 3.3 we apply the stochastic activation function to a standard regression dataset to demonstrate uncertainty quantification.

To aid reproducibility, none of the individual models in this Section took longer than an hour to train on a quad-core laptop. Code to reproduce the results of this Section is available online at https://github.com/tangoed2whiskey/StochasticActivationFunction.

3.1. *Hyperparameter-free self-regularization with MNIST*

MNIST is a standard optical character recognition dataset, consisting of a training set of 60 000 greyscale images of handwritten digits between 0 and 9, with associated labels, and 10 000 test images.[17] The aim is to correctly classify the test set images as one of each of the 10 digits.

We aim to demonstrate the utility of stochastic activation functions by showing how they reduce the complexity of hyperparameter optimization, whilst still enabling accurate model construction. We start with a simple dense neural network architecture, of 784-800-800-10 nodes, as used in the original dropout manuscript.[18] ReLUs were used for the hidden nodes, with linear input and softmax output nodes. This network has many more degrees of freedom than the number of training images, and so is susceptible to overfitting. We train the neural network using stochastic gradient descent, with a fixed learning rate of 0.1, no momentum term, no weight

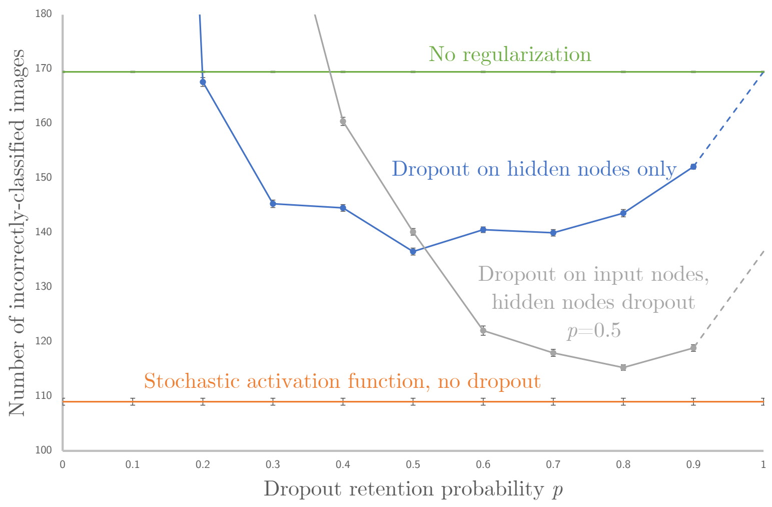

Fig. 1. Number of incorrectly classified images from the MNIST test set, when using no regularization (green line), dropout on the hidden nodes only (blue), dropout on both hidden and input nodes (grey), and the stochastic activation function (orange line). Error bars represent the standard error in the mean, and where not visible are smaller than the size of the points.

regularization, and a cross-entropy loss: this very simple configuration prevents confusion between different sources of regularization.

The state-of-the-art for dense neural networks without any form of regularization or data augmentation achieves around 160 errors out the 10 000 test images. Our very simple network, trained for just 250 epochs, achieves an error rate of 169.5(1), averaged over the last 50 epochs of training, which is shown in green in Fig. 1.

Adding dropout to the hidden nodes of this neural network reduces the number of incorrectly-classified images, if the dropout rate p is set correctly. For $p \geq 0.2$ the dropout network is more accurate than the un-regularized network, with peak accuracy achieved around $p = 0.5$. This is shown in blue in Fig. 1. Adding dropout to the input nodes as well improves the accuracy further: however, p must be set higher for the input nodes, around $p = 0.8$: lower values reduce the accuracy relative to not using dropout at all. In Fig. 1, the grey line shows the effect of including dropout on the input nodes, with a fixed rate of $p = 0.5$ for the hidden nodes. We note that these values of p need to be set empirically, through a hyperparameter optimization stage, and the resulting accuracy is highly dependent on the value of p so selected: naïvely setting the dropout rate for the input nodes to the same as the hidden nodes, $p = 0.5$, results in worse accuracy than not using dropout for the input nodes at all.

In contrast, neglecting dropout and simply replacing all the input and hidden node activation functions in the network with stochastic activation functions generates just 109.0(6) errors out of the 10 000 test set images, shown in orange in

Fig. 1. The stochastic activation functions were applied in exactly the same way to the input and hidden nodes, simply multiplying each node output by random numbers drawn from identical uniform distributions, and required no hyperparameter optimization, achieving accuracy with this neural network architecture of better than a fully-optimized dropout network. This ease of application and competitive accuracy are key benefits of the proposed stochastic activation function.

Stochastic activation functions may of course be combined with other forms of regularization, including standard L1/L2 regularization, weight decay, weight norm clipping, and data augmentation. As both stochastic activation functions and dropout implement sampling from larger spaces of models, it is not recommended to combine both methods on the same node.

3.2. *Feature simplification with CIFAR-10*

CIFAR-10 is a set of 32×32 color images representing ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There are 50 000 training images, with the aim being to correctly classify each of 10 000 test images.[19] As there are obvious similarities and even subjective overlap between classes, this is a more difficult machine vision task than classifying the MNIST images. To make progress, we start from a version of the VGG19 convolutional neural network[20] with the convolutional weights pre-trained on ImageNet,[21] available through Keras.[22] We freeze these pre-trained layers, remove the top dense, classification layers, and replace them with three hidden layers of 256 ReLUs and a final softmax classification layer which we train on the CIFAR-10 data.

Again this network architecture is susceptible to overfitting. To emphasize the regularization effects of stochastic activation functions we push the network further into the overfitting regime with a high initial learning rate of 5×10^{-3} using the Adam optimizer[23] and do not initially use any forms of regularization.

Figure 2 shows the test-set accuracy over the training epochs using the unregularized neural network in green. It is apparent that there is significant overfitting occurring, as after initially rising the accuracy drops over the training epochs before the weights settle into a local optimum with 56.7(4)% of the images correctly classified. This overfitting behavior may be mitigated by using regularization techniques: using L1 regularization, adding a penalty term to the magnitude of the weights, stops the overfitting behavior, shown in blue in Fig. 2, achieving higher accuracy with 58.6(6)% of the images correctly classified, but at the cost of increased variance in the accuracy of predictions. Here the L1 term coefficient was optimized over the range 1×10^{-5} to 1×10^{-3}, with final value 5×10^{-4}.

Dropout also provides regularization; here with $p = 0.5$ on the unfrozen hidden nodes and $p = 0.8$ on the "input" nodes from the convolutional layers, which results in an accuracy curve that does not overfit, shown in grey in Fig. 2, with an accuracy of 57.9(3)%. Similarly, replacing all the unfrozen hidden layer nodes with stochastic activation functions, based on underlying linear units for the "input" nodes and

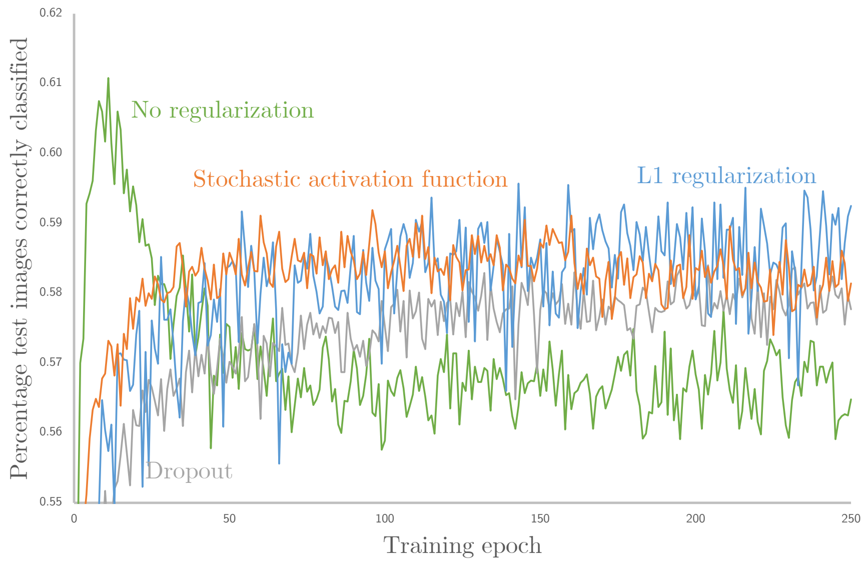

Fig. 2. Accuracy of predictions on the CIFAR-10 test set as a function of training epoch, showing the unregularized neural network result in green, a network regularized using dropout in grey, a network using L1 regularization in blue, and a network that uses stochastic activation functions in orange.

ReLUs for the hidden nodes, strongly reduces the overfitting of the network, as shown in orange in Fig. 2, and achieves an accuracy of 58.2(3)%. The reduced variance in accuracy of predictions from the stochastic activation function relative to the unregularized network and, especially, the network using L1 regularization is an important benefit in situations where consistency of results is important, including for example medical applications.

Importantly, the stochastic activation functions allow the network to train quicker than dropout, with a steeper initial curve of accuracy in Fig. 2 than either of the other methods of regularization. With the computational expense of modern deep learning approaches,[24] methods that mitigate against training time, as well as providing regularization and reduced hyperparameter optimization complexity, could make a valuable addition to the machine learning toolbox.

One key result of including regularization in a neural network is the sparsification of the node activations. This reduces the complexity of the model being learnt, and enables increases in the efficiency of training and predictions by leveraging the power of sparse matrix algebra. We demonstrate this sparsification in Fig. 3, where the mean activations across the test set for each of the $3 \times 256 = 768$ hidden nodes are plotted. The unregularized network has an average activation of 0.85(3), compared to 0.163(6) when using dropout and 0.191(7) when using stochastic activation functions. This indicates that the stochastic activation functions have had the effect of reducing the size of the node activations, encouraging the learning of

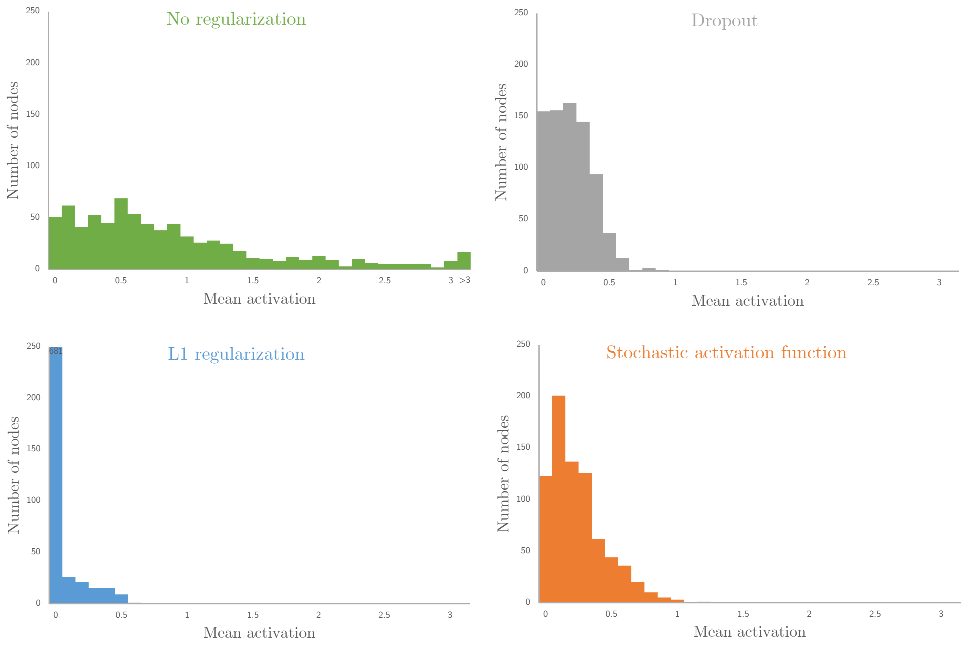

Fig. 3. Histograms of the mean activation over the 768 hidden nodes of the neural networks, with the result for the unregularized network on the top left, the network using dropout on the top right, the network using L1 regularization on the bottom left, and the network using stochastic activation functions on the bottom right.

simpler representations. The sparsification is not as extreme as that imposed by L1 regularization, where 681 of the 768 hidden nodes have average activation below 0.1, with average activation of only 0.023(3). This level of sparsification may limit the complexity of the representations that can be learnt by the network, as indicated by the higher uncertainty in the accuracy on the test set in Fig. 2.

3.3. *Uncertainty quantification with Boston housing data*

For applicability to domains where confidence in results is vital, including healthcare, pharmaceutical, and financial applications, some measure of the uncertainty in predictions generated by deep learning methods is vital. Here we examine the ability of stochastic activation functions to generate accurate estimates of the uncertainty in the predictions of a neural network, using a public dataset on house prices in Boston in the late 1970s.[25] This widely-used dataset provides 13 feature variables for each of 506 towns, with the aim being to predict the median value of the house prices in each town. The average of the median house prices across all 506 towns is around \$22 500.

We generated a random holdout test set of 20% of the towns, and trained a dense neural network with two hidden layers of 800 hidden nodes each on the remaining 80% of the towns. Using no regularization, and training for 100 epochs, this network achieved a root mean square deviation in predicted median house price of around $2600 across the 102 test set towns. We also trained networks using the same architecture and training method, but including dropout and, separately, stochastic activation functions on the hidden layers. As the input features are very information-dense we did not include dropout or stochastic activation functions on the input nodes.

Both dropout and the stochastic activation functions provided comparable accuracy on the test set to not using any regularization. However, these methods both provide access to estimates of the confidence in each prediction. This is achieved by recalling that both methods effectively provide a sampling from a large space of neural networks, which either share weights (dropout) or architecture (stochastic activation functions). We generate estimates of the uncertainty in predictions for each town's median house price by making 100 predictions for each price, in two different ways:

Explicitly. We generate 100 random samples of networks, by applying either dropout or stochastic activation functions at test time as well as train time. The predicted value is then taken as the mean of these predictions, and the uncertainty as the standard deviation in them.

Implicitly. We generate one prediction using the networks without dropout or stochastic activation functions activated at test time (replacing $\mathcal{S}$ with 1 in Eq. (1)), which we use as the predicted value, and then separately take the standard deviation of 99 other networks with dropout or the stochastic activation functions activated to provide an uncertainty estimate.

The implicit sampling method might be expected to generate slightly better mean predictions than the explicit method, because it implicitly samples from a much larger space of possible networks than the explicit sampling can provide, although when many explicit samples are taken the results would be expected to coincide.

We test the accuracy of the uncertainties in the predictions by sorting the predictions in descending order of their uncertainties, and comparing only those with the lowest uncertainties against the test set. This procedure focusses on only the predictions that the neural network expects to be most accurate: the results are shown in Fig. 4 for a range of different percentages of predictions included. We note that this procedure does not use any knowledge of the test set to choose which predictions to include, and hence is perfectly statistically valid.

As we move left in Fig. 4 the number of towns that predictions are made for decreases: but Fig. 4 also shows that, when using either dropout or stochastic activation functions, the deviation between true and predicted values reduces as the least confident predictions are discarded. This indicates that the confidence estimates are accurate, in the sense of being able to identify the most and least confident predictions. By the point only a single town's median house price is being

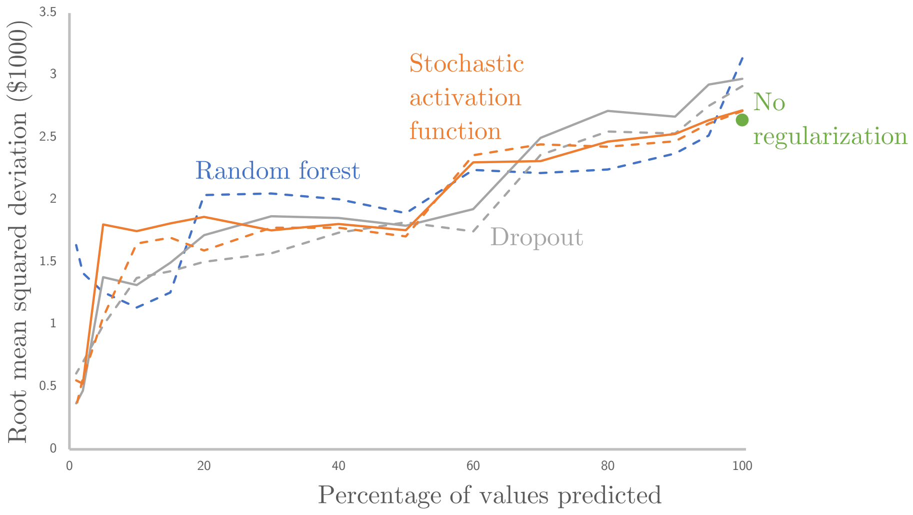

Fig. 4. Root mean square deviation in predicted median house price for the towns in the holdout test set, showing the increase in accuracy as only the most confident predictions are retained. An unregularized network is shown as a green point, as it does not provide a criterion to choose the most confident predictions; networks regularized using dropout and stochastic activation units are shown in grey and orange respectively, with the solid (dashed) lines indicating implicit (explicit) samples from the space of networks. The blue dashed line shows predictions made using a random forest algorithm.

predicted, the root mean squared deviation has dropped to a fifth of the deviation across all towns. The method used for generating the predictions and uncertainties, implicit or explicit sampling (shown as solid and dashed lines respectively) makes little difference to the accuracy achieved in this small dataset.

In Fig. 4 we also show predictions made using an ensemble method, a regression random forest, which explicitly trains a sample of (here 100) decision trees from the space of possible decision trees on the training data.[26,27] This method is designed for exactly the type of regression problem presented by this dataset and provides comparable accuracy to the neural networks, including in its selection of the most confident predictions. However, random forests are not generalizable to problems such as image or speech recognition, where deep neural networks using dropout or stochastic activation functions may still provide accurate confidence estimates.

The scale of the standard deviation uncertainties generated by the uncertainty quantification measures examined here are all approximately consistent with the predictions being normally distributed around the true values (which would suggest 68% of the predictions should be within one standard deviation of the true values, for example). These relationships could be made exact by applying conformal prediction to the generated uncertainty estimates,[28] but this will not modify the relative ordering of the predictions when sorted by uncertainty, and hence will not affect the results of Fig. 4.

4. Conclusions

Stochastic activation functions provide the same benefits as dropout, in terms of regularization arising from the sampling of a large space of networks, but are hyperparameter-free and hence provide easier training and interpretation. Stochastic activation functions have been shown to provide efficient regularization, sparsification, and uncertainty quantification on a variety of publically-available benchmark sets.

Stochastic activation functions are applicable to all standard nodes in artificial neural networks, except where the amplitude of the response is required to be a fixed number (e.g. skip connections in ResNet, as in Ref. 29). It would also be possible to extend the application of stochastic activation functions to convolutional and recurrent layers.

The application of stochastic activation functions beyond linear and ReLU functions is also an avenue for future research: the uniform random sampling defined in Eq. (2) for stochastic activation functions is well-suited to these linear and piecewise-linear functions, but other distributions might be more appropriate for other activation functions. The concept of the activation function as a cumulative probability distribution might inform the selection of the sampling distribution.

Code implementing the stochastic activation function is freely available online at https://github.com/tangoed2whiskey/StochasticActivationFunction.

Acknowledgments

I acknowledge the support of my colleagues at Intellegens, particularly Ben Pellegrini and Gareth Conduit.

References

1. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine* **29**(6) (2012a) 82–97, doi:10.1109/MSP.2012.2205597.
2. Y. Guo, Y. Liu, T. Georgiou and M. S. Lew, A review of semantic segmentation using deep neural networks, *International Journal of Multimedia Information Retrieval* **7**(2) (2018) 87–93, doi:10.1007/s13735-017-0141-z.
3. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, *Science* **362**(6419) (2018) 1140–1144, doi:10.1126/science.aar6404.
4. G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems* **2**(4) (1989) 303–314, doi:10.1007/BF02551274.
5. A. Y. Ng, Feature selection, L1 vs. L2 regularization, and rotational invariance, in *Proc. of the Twenty-first Int. Conf. on Machine Learning* (*ICML '04*) (ACM, New York, NY, USA, 2004), pp. 78–87, doi:10.1145/1015330.1015435.
6. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research* **15** (2014) 1929–1958, http://jmlr.org/papers/v15/srivastava14a.html.

7. L. Perez and J. Wang, The effectiveness of data augmentation in image classification using deep learning, *CoRR*, abs/1712.04621 (2017), http://arxiv.org/abs/1712.04621.
8. D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Nature* **323**(6088) (1986) 533–536, doi:10.1038/323533a0.
9. G. Klambauer, T. Unterthiner, A. Mayr and S. Hochreiter, Self-normalizing neural networks, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Curran Associates, Inc., 2017), pp. 971–980, http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf.
10. V. Nair and G. E. Hinton, Rectified linear units improve restricted Boltzmann machines, in *Proc. of the 27th Int. Conf. on Int. Conf. on Machine Learning* (*ICML'10*) (Omnipress, USA, 2010), pp. 807–814, http://dl.acm.org/citation.cfm?id=3104322.3104425.
11. R. M. Zur, Y. Jiang, L. L. Pesce and K. Drukker, Noise injection for training artificial neural networks: A comparison with weight decay and early stopping, *Medical Physics* **36**(10) (2009) 4810–4818, doi:10.1118/1.3213517.
12. J. Lampinen and A. Vehtari, Bayesian approach for neural networks — Review and case studies, *Neural Networks* **14**(3) (2001) 257–274, doi:10.1016/S0893-6080(00)00098-8.
13. U. Naftaly, N. Intrator and D. Horn, Optimal ensemble averaging of neural networks, *Network: Computation in Neural Systems* **8**(3) (1997) 283–296, doi:10.1088/0954-898X_8_3_004.
14. H. M. D. Kabir, A. Khosravi, M. A. Hosen and S. Nahavandi, Neural network-based uncertainty quantification: A survey of methodologies and applications, *IEEE Access* **6** (2018) 36218–36234.
15. B. D. Conduit, N. G. Jones, H. J. Stone and G. J. Conduit, Design of a nickel-base superalloy using a neural network, *Materials & Design* **131** (2017) 358–365, doi:10.1016/j.matdes.2017.06.007.
16. T. M. Whitehead, B. W. J. Irwin, P. Hunt, M. D. Segall and G. J. Conduit, Imputation of assay bioactivity data using deep learning, *Journal of Chemical Information and Modeling* **59**(3) (2019) 1197–1204, doi:10.1021/acs.jcim.8b00768.
17. Y. LeCun, C. Cortes and C. J. C. Burges, The MNIST database, http://yann.lecun.com/exdb/mnist/.
18. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, *arXiv e-prints*, art. arXiv:1207.0580 (2012b).
19. A. Krizhevsky, Learning multiple layers of features from tiny images, Technical Report, University of Toronto (2009).
20. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv e-prints*, art. arXiv:1409.1556 (2014).
21. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and F.-F. Li, ImageNet: A large-scale hierarchical image database, in *CVPR09* (2009).
22. F. Chollet *et al.*, Keras, https://keras.io (2015).
23. D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, *arXiv e-prints*, art. arXiv:1412.6980 (2014).
24. E. Strubell, A. Ganesh and A. McCallum, Energy and policy considerations for deep learning in NLP, *arXiv e-prints*, art. arXiv:1906.02243 (2019).
25. D. Harrison and D. L. Rubinfeld, Hedonic housing prices and the demand for clean air, *Journal of Environmental Economics and Management* **5**(1) (1978) 81–102, doi:10.1016/0095-0696(78)90006-2.

26. L. Breiman, Random forests, *Machine Learning* **45**(1) (2001) 5–32, doi:10.1023/A:1010933404324.
27. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research* **12** (2011) 2825–2830.
28. I. Cortés-Ciriano and A. Bender, Reliable prediction errors for deep neural networks using test-time dropout, *Journal of Chemical Information and Modeling* **59** (7) (2019) 3330–3339, doi:10.1021/acs.jcim.9b00297.
29. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conf. on Computer Vision and Pattern Recognition* (*CVPR*) (2016), pp. 770–778, doi:10.1109/CVPR.2016.90.