Enhanced Dense Layers Using a Quadratic Transformation Function

Atharva Gundawar and Srishti Lodha

1 Introduction

Dense layers, consisting of neurons, are the building blocks of any deep learning architecture. Neurons compute output as a sum of the following two features: (1) the summation of the product of the output vectors or logits from the previous layers with the corresponding trainable weight vector and (2) a trainable linear bias (refer Fig. 1).

Here, the function "f" refers to the selected activation function, "b" refers to the bias, "x_i" refers to input to the neuron, and "w_i" refers to the weight assigned to that neuron. X and Y are the input and output vectors of the neuron, respectively.

The numerous developments in deep neural network architectures, including techniques like dropout [1] and pruning, have helped overcome problems like exploding gradients and biased graphs. Some models involve skip connections (e.g., ResNet [2]), while some contain parallel paths (like InceptionNet [3]). While the difference between these models lies in the arrangement of layers, connections, paths traced by the logits, and so on, the underlying transformation function still remains the same. Hence, by changing this computation, every neuron reflects a minor change. Combined, all the neurons greatly impact the final result of the model.

Numerous mathematical functions can be explored to replace the linear function that calculates the output of a neuron. Within the scope of this paper, we build and test a simple dense neural network with a quadratic transformation function. The output is now a sum of the summation of the products between the corresponding trainable weight vectors firstly with the input and secondly with the input squared, and the trainable linear bias. This equation is presented in Fig. 2 ("w_i" and "w_j" refer to the weights assigned to that neuron.). We acquired four popular datasets to train

A. Gundawar · S. Lodha (✉)
Vellore Institute of Technology, Vellore, Tamil Nadu, India

A. Haldorai et al. (eds.), *5th EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing*, EAI/Springer Innovations in Communication and Computing, https://doi.org/10.1007/978-3-031-28324-6_1

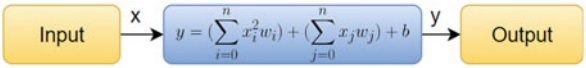

Fig. 1 Linear transformation function in the conventional perceptron

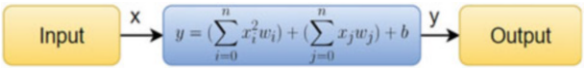

Fig. 2 Quadratic transformation function in the proposed perceptron

and evaluate the model. On comparing the results that conventional neurons [4] and new neurons yielded from a simple architecture with convolutional and dense layers, we observe an improved accuracy on any fixed number of epochs. Analysis of this improvement produces notable results, which are discussed later in this study.

The remaining paper is organized as follows. We conduct a literature review of closely related research and highlight their major contributions and drawbacks (Sect. 2). In Sect. 3, we discuss in depth, the methodology adopted for the implementation of this study. Finally, we move to the results and analysis in Sect. 4, followed by the conclusion (Sect. 5) and the references.

2 Literature Review

In this section, several related studies have been reviewed, and their contributions have been highlighted. We also identified certain drawbacks in these papers.

In H. Lin et al. [5], a universal approximation method was implemented by copying the Resnet structure, but with only one hidden neuron in alternating dense layers. This neuron presented a very high-order function, which was a representation of the combination of all neurons from a conventional hidden dense layer. Over-parametrization was successfully reduced and a universal approximation theorem for Resnet was implemented. However, this approach does not perform better than the pre-existing Resnet architecture in terms of accuracy in classification tasks. In F. Fan et al. [6], a successful autoencoder architecture was made using convolutional layers and quadratic dense functions, which replace the traditional single-order dense layers. The paper achieved the best results numerically and clinically in the dataset cited in the paper. As this works very well on the targeted dataset, we have no information if these results are translatable to other datasets and architectures as well.

V. Kůrková et al. [7] used shallow signum perceptron to achieve a lower bound on errors in approximation. In this probabilistic approach, the authors have shown that lower bounds on errors can be derived from the total number of neurons on finite domains. Not only is the proposal restrictive to certain domains, but unless a minimum threshold of signum perceptron is present in a given layer, the

approximation of errors in both binary-valued and real-valued functions fails to give reliable results. In another study, C. L. Giles and Y. Maxwell [8] presented a new system of storing knowledge in higher-order neural networks with the use of priori knowledge. This system resulted in a model which attained a higher accuracy on multiple datasets and handled outliers better than the models used in the comparative study. However, having a priori knowledge system for any dataset in the real world is improbable, and the increase in accuracies does not always make up for the high model size.

S. Du and J. Lee [9] found a strong relationship between the number of hidden nodes activated by a quadratic function and the number of training examples. The theory of Rademacher complex was used to show how a trained model generalizes. Further research in terms of how the local search algorithm using over-parametrization finds very close global minimas can be conducted.

Similar to our study, F. Fan et al. [10] introduced a new type of neuron to replace the original dense layer neuron. Although it also has an order of 2, the function is a sum of two terms entirely different from what is proposed in this study. Instead, it is a summation of: (a) the product of the outputs of the conventional transformation function with a different set of weights for the same input "x" and (b) the output of the conventional transformation function but with the input squared. These 2nd order neurons worked well in solving low complexity tasks, like fuzzy logic problems, and representing basic logic gates like "and," "or," "nand," and "nor," but the research fails to put some light on the working of this principle on multi-layer neural networks. The implementation in this research was confined to testing the working theory on a single perceptron, with the aim of building a perceptron capable of learning a more complex function than the simple linear function. More research and analysis has to be done on multi-layer Neural Network (NN) and deep NN architectures, where the results are compared to the conventional transformation function. F. Fan et al. [11] also introduced a backpropagation algorithm especially to better pass the gradients in the backward pass to update the weights of a second-order neural network. There is a significant change in the accuracy of the models used for comparison in this study; one trained using the traditional backpropagation and the other trained using the new backpropagation algorithm discussed in this paper. However, the paper summarizes results only from benchmarked biomedical and engineering datasets and hence is not enough to prove its working in real-world datasets.

In M. Blondel [12], training algorithms of HOFMs (higher-order factorization machines) [12] and new HOFMs with new formulas that used shared parameters have been presented. The study does a good job in terms of exploration of different functions and augmentations of HOFMs which can be applied to neurons. While the results have proven to be quite significant, the depth of the algorithms was not much. Some training algorithms, if not most, have a lot of scope for fine-tuning.

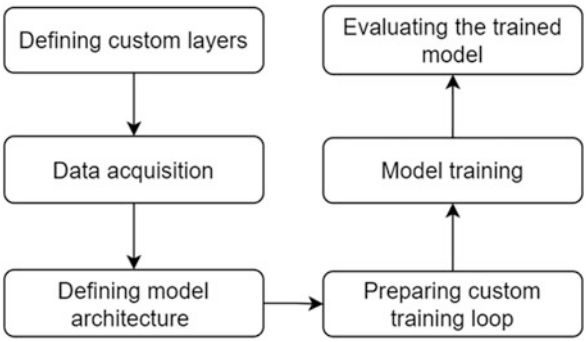

Fig. 3 Summarized process pipeline

3 Methodology

The objective of this research is to propose an underlying architecture of dense layers that improves the performance of all deep-learning models. This section contains details of the entire research pipeline, starting from the custom layer definition to the evaluation of the trained model. A simple convolutional neural network is used to test the proposed architecture on five different well-known datasets. The process pipeline is summarized in Fig. 3.

3.1 Defining Custom Layers

To define a new layer, we implement three main steps. These include the layer definition, defining the type of variable and the computational kernel used, and implementing a call function, where we define the forward pass logic of the neuron.

For initialization, we define the units and the activation. We can control the number of units, the type of units, and other input independent initializations required when we build the model. All constant and non-constant variables (which directly or indirectly affect the computation in the forward pass) are defined here and added to the class scope for access by other class functions. The building section of the layer definition consists of defining the scope and the datatypes of the variables defined above, as well as a selection of the kernel. This section of the layer is implemented as a part of the model compilation process. During model compilation, memory is allocated for these variables according to their shape and type. Finally, in the section where we implement the calling functionality, the forward pass is coded, where the mathematical function to define what happens to the input variable is described. The function should contain at least the input to the model and return the augmented logit value.

In our research, we initialized the starting parameters using a randomize function which normalized the values for all the weight parameters between 0 and 1 using a

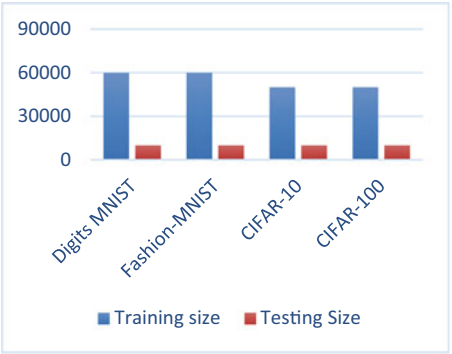

Fig. 4 Training and testing sizes of the datasets used

Gaussian distribution. The bias coefficient, on the other hand, was initialized to 0. Both weight and bias coefficients were 32-bit float values.

After these steps have been completed, a simple test to check the randomness of the initialized values of the defined variables and the working of the forward pass confirms the proper working of the new layer.

3.2 Data Acquisition

As the datasets used in this research are very well maintained by TensorFlow [13], we use its dataset API to acquire pre-structured and organized data (using the load function). The datasets used for this study include 2 MNIST datasets, namely, Handwritten Digits MNIST [14] (containing handwritten digits in grayscale images of numbers from 0 to 9) and Fashion MNIST [15] (containing 10 different classes of clothing items). Both of these consist of 70,000 images, which have a size of 28×28 pixels. The other 2 datasets are the CIFAR-10 [16] and the CIFAR-100 [16] datasets. The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes, and CIFAR-100 with 100 classes (6000 images per class). All these datasets form a benchmark and are recognized by the community for the task of classification. Their training and testing sizes have been shown in Fig. 4. The datasets used have a huge usability index, which is why the data preparation in this research constituted of only rescaling the data and no more augmentation was required.

3.3 Model Architecture and Model Compilation

The model architecture used in this study consists mainly of two types of layers, namely, the convolutional layer and the dense layer. The dense layer can be the conventional or the newly proposed quadratic dense layer. The model consists of

two pairs of convolutional layers followed by a MaxPooling layer and then one single convolutional layer, with the number of filters being 32, 64, 64 from the first layer to the last in that order. The activation for all these layers is ReLU [17], while the input to the first layer is dataset dependent. All the MaxPool layers in these pairs have a kernel size of 2×2. After flattening the output of the last convolutional layer, we have three pairs of the dense layers, followed by a dropout layer, with the number of units for the dense layer being 128, 64, 32 from the first layer to the last in that order. These dense layers are conventional and quadratic respectively in two different research experiments to compare them. The activations for all of these are ReLU and the dropout rate for all the dropout percentages is 20%. Finally, a simple dense layer is added with Softmax activation, and the number of units here equals the number of classes, which is 100 in CIFAR-100 and 10 in the case of the other 3 datasets.

Both the models for all the datasets are compiled with the sparse categorical cross-entropy [18] loss function and the Adam optimizer [19] to handle the gradient. The algorithm for the backpropagation of gradient in the proposed quadratic dense layer can be understood as follows.

Assuming the input variable is $\in R^d$, the intermediate variable can be mathematically represented as:

$$z = W_1^{(1)} x^2 + W_2^{(1)} x \tag{1}$$

where $W_1^{(1)}$, $W_2^{(1)} \in R^{h \times d}$ are the trainable weight parameters. After running the logit or the intermediate variable $z \in R^h$ through the activation function ϕ, we get the hidden activation of the intermediate logit:

$$h = \phi(z) \tag{2}$$

Assuming that the parameters of the output layer only possess a weight of $W^{(2)}1$, $W^{(2)}2 \in R^{q \times h}$, we can obtain an output layer variable with a vector of length q:

$$o = W_1^{(2)} h^2 + W_2^{(2)} h \tag{3}$$

To calculate the loss for a single example, we can denote the loss $\mathbf{L}$ as the value outputted by the loss function for an output h and expected real target label y as:

$$L = l\,(o, y) \tag{4}$$

If we were to introduce $\ell 2$ regularization, then considering the hyperparameter term λ, we can calculate the regularization as:

$$s = \frac{\lambda}{2} \left(\left| 2W_1^{(1)} + W_2^{(1)} \right|_F^2 + \left| 2W_1^{(2)} + W_2^{(2)} \right|_F^2 \right) \tag{5}$$

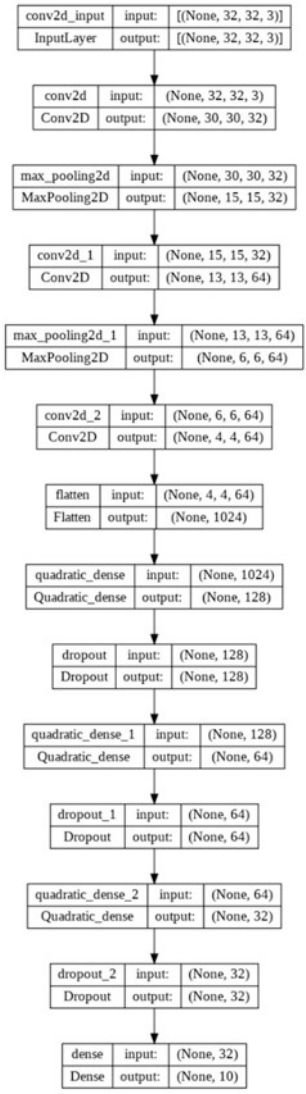

Fig. 5 BERT Model architecture

Now, a customized sequential dense neural network is built (refer Fig. 5). All the connections between blocks and layers, number of units, activation functions, and other parameters to augment the architecture of the model are defined here. The model is compiled with a sparse categorical cross-entropy loss function and a choice of accuracy metrics.

3.4 Preparing Custom Training Loop

Custom training loops include augmentations of what happens in a training step, for example, redefining forward and backward passes while training. When epoch-wise logging the accuracy and losses, early stopping, updating the learning rate with momentum, etc. can be done using callback functions. In our implementation, the patience of the early stopping callback was set to 3, and we monitored the loss. The momentum of the optimizer was set to 0.5. In our implementation, we have used callbacks to record certain accuracies and losses, which are presented in the results section of this paper.

3.5 Model Training and Evaluation

Finally, the model is trained and the losses and accuracies are monitored. TensorBoard [13] has been used in our research to visualize these results.

4 Results and Analysis

As previously mentioned, the model was evaluated on four different datasets. We recorded the conventional accuracy (neurons with linear transformation function) and the accuracy obtained with the proposed model (neurons with quadratic transformation function) at five different epoch values (refer Tables 1 and 2). The results per dataset have also been visualized (Fig. 6). Several notable inferences can be drawn from the results obtained.

The most important inference is that the quadratic dense layer converges much faster than the traditional layers in terms of both training and validation accuracies. For example, take the validation accuracies yielded on CIFAR-100. While the conventional model reaches an accuracy of approximately 26% at the 20th epoch, the new model reaches the same accuracy at around the 10th epoch itself. From the training accuracies on this dataset, we can observe that the linear model yields an accuracy of around 25% after the 20th epoch, while the proposed model reaches the same accuracy in less than five epochs. This shows a performance that is four times better with our model. Moreover, there is always a significant positive difference in the accuracies at every epoch, suggesting that the quadratic function has higher scope of learning. Although a large number of linear neurons can ultimately form any function, including quadratic, directly using this quadratic function in the neuron itself drastically reduces the number of epochs needed to reach a particular accuracy. This in turn reduces the overall computation time. Hence, if a heavy deep learning architecture with conventional neurons can reach the accuracy of 92% in 50 epochs, the same model will produce a 92% accuracy in half (or even fewer) number

Table 1 Training accuracy obtained on varying epochs for four datasets

Dataset	Number of epochs	Accuracy obtained with conventional neurons (%)	Accuracy obtained with quadratic neurons (%)
Handwritten Digits MNIST	1	91.40	91.58
	2	97.82	97.99
	3	98.41	98.47
	4	98.76	98.79
	5	98.96	99
Fashion- MNIST	1	73.12	75.59
	2	84.51	85.63
	3	87.36	88.40
	4	88.86	89.59
	5	89.70	90.18
CIFAR-10	1	32.10	94.19
	2	45.12	94.64
	5	54.41	94.60
	10	73.32	95
	20	81.69	95.70
CIFAR-100	1	2.56	23.34
	2	5.12	24
	5	12.41	25.21
	10	18.49	27.07
	20	24.86	30.26

of epochs with the new neurons. This hypothesis can be successfully validated by implementing the proposed methodology for ResNet, InceptionNet, VGGNet, etc. However, we can predict that both the types of neurons or dense layers might ultimately reach the same accuracy, just with a drastically varying number of epochs.

It is also interesting to note how the initial accuracy yielded by the new neurons is always much better than the old ones. For instance, training and validation accuracies after the first epoch on CIFAR-10 were 94.19% and 90.30% with the proposed neurons, against just 32.10% and 46.10%, respectively, with the conventional ones. On CIFAR-100, this difference is even higher (23.24% against 2.56% while training, and 24.65% against 4.18% during validation). Importantly, the training time was not affected much by the addition of an extra dimension to the neuron's equation. The execution time for every step in all the datasets remained the same (at 5 ms for both the models). Both the models required an average of 7–8 seconds for every epoch to complete. The model weight was affected; the architecture with the quadratic dense layer resulted in a model with 70% more trainable parameters (at 342,468), compared to the conventional dense layer model, which had a total of 201,156 trainable parameters. Both, the total number of

Table 2 Validation accuracy obtained on varying epochs for four datasets

Dataset	Number of epochs	Accuracy obtained with conventional neurons (%)	Accuracy obtained with quadratic neurons (%)
Handwritten Digits MNIST	1	98.10	98.45
	2	98.55	98.87
	3	98.85	98.71
	4	99.02	98.78
	5	98.96	98.98
Fashion- MNIST	1	82.58	82.40
	2	85.51	87.79
	3	88.01	88.83
	4	89.29	89.34
	5	90.12	90.11
CIFAR-10	1	46.10	90.30
	2	56.53	90.46
	5	63.73	90.47
	10	69.52	90
	20	71.20	90.64
CIFAR-100	1	4.18	24.65
	2	8.28	24.89
	5	16.24	25
	10	22.04	25.51
	20	26.47	26.97

parameters and the trainable parameters, stay the same as none of the layers or logits were frozen or untrainable in the model architecture.

Finally, as predicted, the improvement in the accuracy for initial epochs is much higher in CIFAR-100 than in the other datasets. This is because of the large learning potential that produces a high scope for improvement in the accuracy obtained from the first epoch on this dataset, high variance in the data, and the high number of output classes. On the contrary, the first epoch itself is yielding a high accuracy on the MNIST datasets due to a good fit [20]. Hence, the model is easily overfitted, and the results are only recorded at a low number of epochs. Here, we can observe that despite the less room for improvement, the quadratic neurons perform better.

5 Conclusion

This paper proposed a new methodology for a neuron's output computation, by replacing the conventional linear transformation function with a quadratic transformation function. When tested on four different popular datasets for a

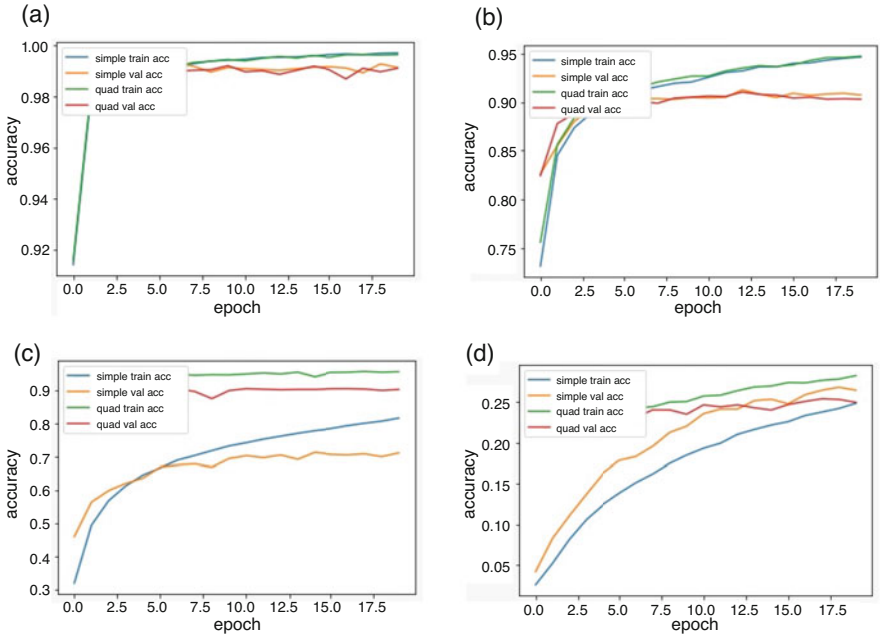

Fig. 6 Simple and quadratic (quad) accuracies plotted for the first 20 epochs for all 4 datasets: (**a**) MNIST, (**b**) Fashion-MNIST, (**c**) CIFAR-10, and (**d**) CIFAR-100

simple dense neural network, an improvement in the accuracy is observed. This improvement is higher when the initial accuracy is low, thus significantly reducing the computation time (and the number of epochs) to arrive at a particular accuracy. Nevertheless, initial convergence to higher accuracies is always much faster in the proposed model. Moreover, the results would become exponentially better with a very large number of neurons in the architecture. The proposed methodology can hence improve the performance of any deep learning architectures containing dense layers.

While the models built using the proposed transformation function do have a higher model weight, they show faster convergence. If we were to increase the number of fully connected layers in the model built using the conventional transformation function with the aim of getting a similar convergence to the quadratic transformation function, we would encounter the vanishing gradient problem. Hence, the proposed methodology also overcomes the vanishing gradient problem in the conventional transformation function when we wish to increase the parameter count, without changing the number of parameters in a given layer.

The real-world applications of these new neurons are numerous. Besides the faster convergence which yields drastically better accuracies when trained for the same duration as conventional neurons, this improved perceptron helps in incredibly reducing the training time of a model in case the same number of parameters are

used in both the models. This in turn means that any research involving DL would be accelerated, and any application utilizing DL models would become more efficient. For instance, in systems like face recognition and real-time threat detection, which use few shot learning techniques, the models will have a lower inference time and yield better accuracies. In the field of medicine, all analyses involving DL would be accelerated, and so on.

There is a vast future scope for this study. While we analyzed the replacement of the linear function of a neuron with a quadratic sum, the same can be replaced with other functions that will potentially yield further improved results [10, 21]. Furthermore, the true power of this methodology can be seen when the results are recorded for larger deep learning architectures, employing a much higher number of parameters. For example, VGG-16 [22], which consists of over 134.7 million parameters, would ideally portray a much better performance on a dataset than ResNet-18, which has 11.4 million parameters.

References

1. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
2. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (Las Vegas, 2016), pp. 770–778
3. M. Lin, Q. Chen, S. Yan, Network in network. arXiv preprint arXiv:1312.4400 (2013)
4. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386 (1958)
5. H. Lin, S. Jegelka, Resnet with one-neuron hidden layers is a universal approximator. Adv. Neural Inf. Proces. Syst. **31** (2018)
6. F. Fan, H. Shan, M.K. Kalra, R. Singh, G. Qian, M. Getzin, Y. Teng, J. Hahn, G. Wang, Quadratic autoencoder (Q-AE) for low-dose CT denoising. IEEE Trans. Med. Imaging **39**(6), 2035–2050 (2019)
7. V. Kůrková, M. Sanguineti, Probabilistic lower bounds for approximation by shallow perceptron networks. Neural Netw. **91**, 34–41 (2017)
8. C.L. Giles, T. Maxwell, Learning, invariance, and generalization in high-order neural networks. Appl. Opt. **26**(23), 4972–4978 (1987)
9. S. Du, J. Lee, On the power of over-parametrization in neural networks with quadratic activation, in *International Conference on Machine Learning*, (PMLR, 2018), pp. 1329–1338
10. F. Fan, W. Cong, G. Wang, A new type of neuron for machine learning. Int. J. Numer. Method Biomed. Eng. **34**(2), e2920 (2018)
11. F. Fan, W. Cong, G. Wang, Generalized backpropagation algorithm for training second-order neural networks. Int. J. Numer. Method Biomed. Eng. **34**(5), e2956 (2018)
12. M. Blondel, A. Fujino, N. Ueda, M. Ishihata, Higher-order factorization machines. Adv. Neural Inf. Proces. Syst. **29** (2016)
13. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., *TensorFlow: A System for Large-Scale Machine Learning* (OSDI, 2016), pp. 265–283
14. L. Deng, The MNIST database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)

15. H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
16. A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images* (University of Toronto, Toronto, 2009)
17. A.F. Agarap, Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375 (2018)
18. J. Bridle, Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. Adv. Neural Inf. Proces. Syst. **2** (1989)
19. D.P. Kingma, J. Ba, Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. N. Venkat, *The Curse of Dimensionality: Inside Out* (2018). https://doi.org/10.13140/RG.2.2.29631.36006
21. F. Fan, J. Xiong, G. Wang, Universal approximation with quadratic deep networks. Neural Netw. **124**, 383–392 (2020)
22. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)