Qi and Wang
Visual Computing for Industry, Biomedicine, and Art (2022) 5:23
https://doi.org/10.1186/s42492-022-00118-z

Visual Computing for Industry, Biomedicine, and Art

ORIGINAL ARTICLE

Open Access

Superiority of quadratic over conventional neural networks for classification of gaussian mixture data

Tianrui Qi and Ge Wang*

Abstract

To enrich the diversity of artificial neurons, a type of quadratic neurons was proposed previously, where the inner product of inputs and weights is replaced by a quadratic operation. In this paper, we demonstrate the superiority of such quadratic neurons over conventional counterparts. For this purpose, we train such quadratic neural networks using an adapted backpropagation algorithm and perform a systematic comparison between quadratic and conventional neural networks for classificaiton of Gaussian mixture data, which is one of the most important machine learning tasks. Our results show that quadratic neural networks enjoy remarkably better efficacy and efficiency than conventional neural networks in this context, and potentially extendable to other relevant applications.

Keywords: Artificial neural networks, Quadratic neurons, Quadratic neural networks, Backpropagation, Classification, Gaussian mixture models

Introduction

In machine learning, the mainstream approach is now artificial neural networks (ANNs), especially deep neural networks. Usually, a neural network consists of several layers of neurons, each of which consists of a linear compartment in the form of the inner product of inputs and weights and a nonlinear unit known as an activation function to make a signal on (activated) or off (attenuated). Deep neural networks have been recently shown to achieve remarkable successes in various applications such as natural language processing [1, 2], auto-driving [3–5], game-playing [6], image analysis [7, 8], and image reconstruction [9].

Classification/clustering is one of the essential pattern recognition techniques in machine learning, and has a wide arrange of applications such as bioinformatics [10, 11] and medial imaging [12, 13]. It is well known that the Gaussian mixture model (GMM) is the most popular data model. Since the prior probability of each Gaussian component is typically not given, known as latent variables, the correct parameters of GMM are solved using the expectation-maximization (EM) algorithm. Alternatively, a neural network approach can be used to classify GMM data. Clearly, the decision boundary for the classification can be viewed as a complicated function where a network with a large number of neurons can approximate that boundary. After the classification network is trained, the inference by the trained network is more efficient than the EM algorithm, which is iterative and time-consuming.

In our previous study [14–19], a new type of neurons, referred to as quadratic neurons, was introduced, where the inner product inside a conventional neuron is upgraded to a quadratic function. The initial motivation is to enrich the diversity of artificial neurons, inspired by the fact that the biological diversity exists at the cellular level, and such diversity enables efficiency, flexibility, functionality, and other benefits. Hence, it is hypothesized that a quadratic neural network would be advantageous similarly, which can, for example, approximate

*Correspondence: wangg6@rpi.edu

Department of Computer Science, Rensselaer Polytechnic Institute, NY 12180, Troy, USA

a given function with a lighter structure than a conventional neural network.

The main purpose of this paper is to highlight the superiority of quadratic over conventional neural networks with the classification task as an illustrative example. The rest of the paper is organized as follows. In the next section, we review the theoretical minimum error in the GMM classification and the EM algorithm that is traditionally used to reach that error bound. In the third section, we present our procedure for initializing and training the conventional and quadratic networks with an adapted backpropagation (BP) algorithm. In the fourth section, we perform numerical experiments systematically and establish the superiority of quadratic networks over the conventional counterparts in the GMM classification. Finally, in the last section we discuss relevant issues and conclude the paper.

Methods

GMM-based classification error

In statistical classification, the Bayes error rate is theoretically optimal. In practice, without knowing latent GMM parameters, the Bayes error rate cannot be directly calculated. To close the gap, the classic EM algorithm can be used to approximate the optimal error rate, which is the benchmark to evaluate the performance of classification neural networks.

Bayes error

Given the mean μ, covariance $\mathbf{C}$, and prior probability π of each Gaussian component of GMM $\mathcal{N}$, the posterior probability $p(z_k = 1|x_n)$ is calculated by

$$p(z_k = 1|x_n) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \mathbf{C}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x_n|\mu_i, \mathbf{C}_i)} \tag{1}$$

which means a D dimensional sample vector x_n, $n \in \{1, \ldots, N\}$, should be assigned to the $\hat{\mathbf{y}}x_n$ Gaussian component,

$$\hat{\mathbf{y}}_n = \arg \max_{k \in \{1,\ldots,K\}} p(z_k = 1|x_n) \tag{2}$$

where D, N, and K represent the dimensionality of the sample vector, the sample size and the number of Gaussian components respectively. We can obtain the Bayesian inference results, a size N vector $\hat{\mathbf{y}}$, by applying Eq. 2 to the entire sample pool $\mathbf{x} = [x_1, \ldots, x_N]^T$ and compare it with the ground truth labels $\mathbf{t}$. However, in most of real cases all these GMM parameters are not directly known. Fortunately, we can use the EM algorithm to estimate them, as described in the following subsection.

Note that the inference $\hat{\mathbf{y}}$ cannot be directly used as the predicting label of each sample since our task is clustering instead of classification. For example, while the ground truth parameters are $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$, the results from the EM algorithm can be $\hat{\theta} = \{\hat{\theta}_2, \hat{\theta}_4, \hat{\theta}_1, \hat{\theta}_3\}$, $\hat{\theta_k} \approx \theta_k$ for $k \in \{1, 2, 3, 4\}$. Hence, we have to perform an order correction, i.e., rearranging $\hat{\theta}$ as $\{\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3, \hat{\theta}_4\}$.

A solution to this problem is to perform an exclusive search so that the accuracy or loss can be optimized. By doing so, the best match will be found as our final result. More efficiently, the alternating variables method can be used as described in Algorithm 1, a common derivative-free method for numerical optimization, with the idea to maximize the accuracy by exchanging two coordinates each time and fixing all the remaining ones. We set the *MaxCycle* according to the number of Gaussian components K, and in our experiment $MaxCycle = 20$, which is sufficiently large for $K = 8$.

Then, we compute Eq. 2 using parameters after the order correction we present above for the Bayesian inference results $\mathbf{y}$ and gain the Bayes error as the banchmark of performance of neural networks.

Algorithm 1: Alternating variables method for the order correction of the GMM parameters.

Data: $\hat{\theta} = [\hat{\theta}_1, \ldots, \hat{\theta}_K]$ obtained using the EM algorithm where $\hat{\theta}_k = \left\{\hat{\mu}_k, \hat{\mathbf{C}}_k, \hat{\pi}_k\right\}$ for $k \in \{1, \ldots, K\}$, samples $\mathbf{x}$, ground truth labels $\mathbf{t}$, and the max number of cycles $MaxCycle$;

```
ŷ ← Applying Eq. 2 using the current θ̂ ;
MaxAccuracy ← Comparing ŷ and t ;
Cycle ← 0 ;
while Cycle < MaxCycle do
    for i = 0, ..., K do
        for j = 0, ..., K : j ≠ i do
            θ̂[i], θ̂[j] ← θ̂[j], θ̂[i] ;
            ŷ ← Applying Eq. 2 using the current θ̂ ;
            Accuracy ← Comparing ŷ and t ;
            if Accuracy < MaxAccuracy then
                θ̂[i], θ̂[j] ← θ̂[j], θ̂[i] ;
            else
                MaxAccuracy ← Accuracy
            end
        end
    end
    Cycle ← Cycle + 1 ;
end
return θ̂ ;
```

EM algorithm

As a classic iterative method, the EM algorithm consists of the following two steps: expectation (E) and maximization (M). The E step evaluates the expectation function based on the currently available intermediate parameters, and the M step updates the intermediate parameters to maximize the expectation function. To estimate all the $\mu, \mathbf{C}, \pi$ parameters, the expectation function in the E step is the posterior probability $p(z_k = 1|\mathbf{x})$ for $k = 1, \ldots, K$.

To start the EM procedure, for $k = 1, \ldots, K$, we initialize $\mu_k^{(0)}$ with a size D vector that filled with values from the standard normal distribution, $\mathbf{C}_k^{(0)}$ with D by D identity matrix and $\pi_k^{(0)} = 1/K$. Then, for the jth iteration,

$j \geq 0$, the posterior probability in the E step is computed as

$$p^{(j)}(z_k = 1|\mathbf{x}) = \frac{\pi_k^{(j)} \mathcal{N}\left(\mathbf{x}|\mu_k^{(j)}, \mathbf{C}_k^{(j)}\right)}{\sum_{i=1}^{K} \pi_i^{(j)} \mathcal{N}\left(\mathbf{x}|\mu_i^{(j)}, \mathbf{C}_i^{(j)}\right)} \tag{3}$$

in terms of the current parameters $\mu_k^{(j)}, \mathbf{C}_k^{(j)}, \pi_k^{(j)}$, $k = 1, \ldots, K$. After this E step, the M step goes as follows:

$$\begin{aligned}
\mu_k^{(j+1)} &= \frac{\sum_{n=1}^{N} p^{(j)}(z_k = 1|x_n) x_n}{\sum_{n=1}^{N} p^{(j)}(z_k = 1|x_n)} \\
\mathbf{C}_k^{(j+1)} &= \frac{\sum_{n=1}^{N} (x_n - \mu_k) p^{(j)}(z_k = 1|x_n) (x_n - \mu_k)^T}{\sum_{n=1}^{N} p^{(j)}(z_k = 1|x_n)} \\
\pi_k^{(j+1)} &= \frac{1}{N} \sum_{n=1}^{N} p^{(j)}(z_k = 1|x_n)
\end{aligned} \tag{4}$$

The E and M steps are repeated until the parameters being estimated converge within a pre-specified range or a maximum number of iterations is finished. With these estimated GMM parameters, Eq. 2 and Algorithm 1 can be used for GMM-oriented classificaiton.

Neural network training

Training a neural network involves two steps: initialization which sets up network parameters appropriately, and optimization which adjusts the neural parameters iteratively. An optimizer used in the second step is illustrated in Fig. 1. The key idea is to perform computational optimization using the well-known BP algorithm with respect to an objective or loss function.

While the conventional and quadratic neural networks can be trained based on the same idea of computational optimization, they differ in specific steps, since the chain rule must be applied to different functions that summarize data (i.e., inter product versus quadratic operation). Specifically, let us formulate the forward and BP processes in the following two subsections respectively, and then describe the whole process in the third subsection.

Forward computation

An exemplary feed-forward neural network is shown in Fig. 2, including input, hidden, and output layers. There are L layers in total, in each of which there is a number of neurons. A typical layer first implements affine transforms for conventional neurons and quadratic operations for quadratic neurons, and then nonlinear activations $\sigma^{(l)}$ are performed, which are common for conventional and quadratic neurons.

An illustration of the affine layer of conventioinal and quadratic neurons is shown in Fig. 3. For a conventional neural network, the affine transform can be expressed in terms of a input matrix $\mathbf{a}^{(l)}$ and a weight matrix $\mathbf{w}^{(l)}$ plus a bias row vector $\mathbf{b}^{(l)}$ as follows:

$$\mathbf{z}^{(l)} = \mathbf{a}^{(l)} \mathbf{w}^{(l)} + \mathbf{b}^{(l)} \tag{5}$$

For a quadratic neural network, the quadratic transform can be expressed as

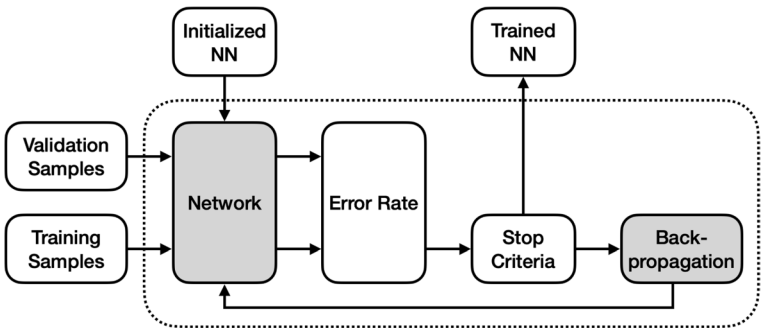

Fig. 1 Neural network training as a computational optimization process with respect to an objective function which is the error rate for classification, without loss of generality

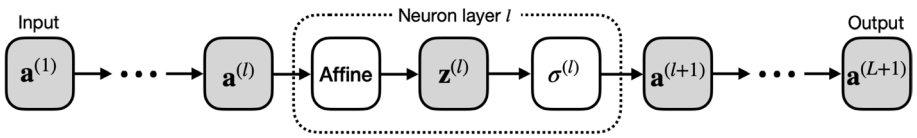

Fig. 2 Forward computation of a feed-forward neural network with L layers of neurons

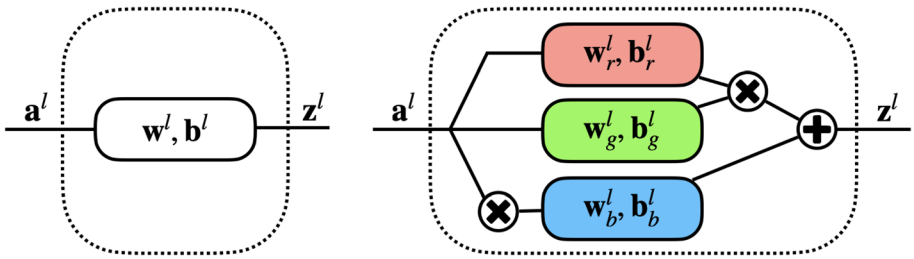

Fig. 3 Illustration of the affine layer of conventioinal (left) and quadratic (right) neurons respectively

$$\mathbf{z}^{(l)} = \left(\mathbf{a}^{(l)}\mathbf{w}_r^{(l)} + \mathbf{b}_r^{(l)}\right) \circ \left(\mathbf{a}^{(l)}\mathbf{w}_g^{(l)} + \mathbf{b}_g^{(l)}\right) + \left(\mathbf{a}^{(l)} \circ \mathbf{a}^{(l)}\right)\mathbf{w}_b^{(l)} + \mathbf{c}^{(l)} \tag{6}$$

where $\mathbf{a}^{(l)}\mathbf{w}^{(l)}$ stands for matrix multiplication and $\circ$ means an element-wise square operation. In this study, the ReLU function is used as the activation function, but if the l-th layer is the last layer of the network, i.e., $l = L$, the softmax function is computed instead. Therefore, the output of each layer is computed as follows:

$$\mathbf{a}^{(l+1)} = \sigma^{(l)}\left(\mathbf{z}^{(l)}\right) = \begin{cases} \max\left(\mathbf{z}^{(l)}, 0\right) & l \neq L \\ e^{\mathbf{z}^{(l)}} / \sum e^{\mathbf{z}^{(l)}} & l = L \end{cases} \tag{7}$$

In other words, the input to the forward process is a N by D sample matrix $\mathbf{a}^{(1)} = \mathbf{x}$, and output is a N by K matrix $\mathbf{a}^{(L+1)}$. The prediction of each sample vector x_n is quantified by

$$\mathbf{y}_n = \arg \max_{k \in \{1,\cdots,K\}} \mathbf{a}_{n,k}^{(L+1)} \tag{8}$$

The loss or error is produced when the prediction differs from the ground truth. Note that in the forward computation we compute and store the output of each affine transform, which are subsequently used for the gradient descent search in the BP process described in the following subsection.

BP formulation

To optimize a neural network, we perform numerical optimization. Specifically, we first find the partial derivatives with respect to each of the parameters and update them via gradient descent search at a suitable step size (learning rate). Using the chain rule, this process was formulated as the well-known BP algorithm, which is widely used to train a neuronal network. As its name indicates, the BP process computes the partial derivatives layer-wise from the output layer to the input layer. A brief BP diagram is shown in Fig. 4.

Let Q stand for the cross-entropy loss value defined as

$$Q = -\sum_{n=1}^{N} \mathbf{t}_n \log \mathbf{y}_n$$

where N is the number of sample vectors, $\mathbf{y}_n$ is the predicted result, and $\mathbf{t}_n$ is the ground truth label for each of the samples x_n. Recall that the activation function of the output layer is the softmax function, hence the gradient of the output layer can be computed as

$$\frac{\partial Q}{\partial \mathbf{z}^{(L)}} = \mathbf{y} - \mathbf{t} \tag{9}$$

If $l \neq L$, the activation function is the ReLU function, and we have

$$\frac{\partial Q}{\partial \mathbf{z}^{(l)}} = \frac{\partial Q}{\partial \mathbf{a}^{(l+1)}} \frac{\partial \mathbf{a}^{(l+1)}}{\partial \mathbf{z}^{(l)}} = \begin{cases} \partial Q / \partial \mathbf{a}^{(l+1)} & \mathbf{a}^{(l+1)} > 0 \\ 0 & \mathbf{a}^{(l+1)} \leq 0 \end{cases}, \qquad l \neq L \tag{10}$$

For a conventional neural network, we know that

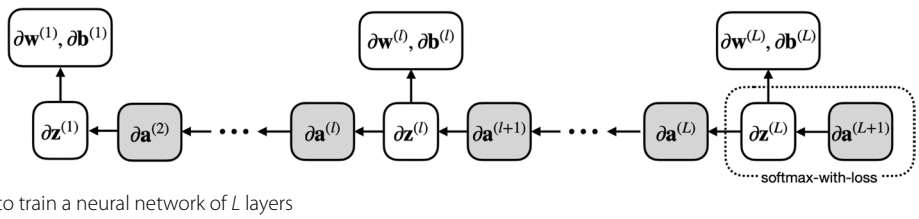

Fig. 4 BP process to train a neural network of L layers

$$\begin{aligned}\frac{\partial Q}{\partial \mathbf{w}^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{w}^{(l)}} = \left(\mathbf{a}^{(l)}\right)^T \frac{\partial Q}{\partial \mathbf{z}^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{b}^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \sum \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\end{aligned} \tag{11}$$

where

$$\frac{\partial Q}{\partial \mathbf{a}^{(l)}} = \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{a}^{(l)}} = \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\left(\mathbf{w}^{(l)}\right)^T \tag{12}$$

The same chain rule can be applied to optimize a quadratic neural network layer-wise. Specifically, let us consider Eq. 6 in the following three parts:

$$\begin{aligned}\mathbf{z}^{(l)} = &\underbrace{\left(\mathbf{a}^{(l)}\mathbf{w}_r^{(l)} + \mathbf{b}_r^{(l)}\right)}_{\mathbf{z}_r^{(l)}} \circ \underbrace{\left(\mathbf{a}^{(l)}\mathbf{w}_g^{(l)} + \mathbf{b}_g^{(l)}\right)}_{\mathbf{z}_g^{(l)}} \\ &+ \underbrace{\left(\mathbf{a}^{(l)} \circ \mathbf{a}^{(l)}\right)\mathbf{w}_b^{(l)} + \mathbf{c}^{(l)}}_{\mathbf{z}_b^{(l)}}\end{aligned}$$

and we have

$$\begin{aligned}\frac{\partial Q}{\partial \mathbf{z}_r^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{z}_r^{(l)}} = \mathbf{z}_g^{(l)}\frac{\partial Q}{\partial \mathbf{z}^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{z}_g^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{z}_g^{(l)}} = \mathbf{z}_r^{(l)}\frac{\partial Q}{\partial \mathbf{z}^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{z}_b^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{z}_b^{(l)}} = \frac{\partial Q}{\partial \mathbf{z}^{(l)}}\end{aligned}$$

Then, the gradients with respect to the parameters in the three parts can be respectively found as follows:

$$\begin{aligned}\frac{\partial Q}{\partial \mathbf{w}_r^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}}\frac{\partial \mathbf{z}_r^{(l)}}{\partial \mathbf{w}_r^{(l)}} = \left(\mathbf{a}^{(l)}\right)^T \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{b}_r^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}}\frac{\partial \mathbf{z}_r^{(l)}}{\partial \mathbf{b}_r^{(l)}} = \sum \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{w}_g^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_g^{(l)}}\frac{\partial \mathbf{z}_g^{(l)}}{\partial \mathbf{w}_g^{(l)}} = \left(\mathbf{a}^{(l)}\right)^T \frac{\partial Q}{\partial \mathbf{z}_g^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{b}_g^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}}\frac{\partial \mathbf{z}_r^{(l)}}{\partial \mathbf{b}_g^{(l)}} = \sum \frac{\partial Q}{\partial \mathbf{z}_g^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{w}_b^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_b^{(l)}}\frac{\partial \mathbf{z}_b^{(l)}}{\partial \mathbf{w}_b^{(l)}} = \left(\mathbf{a}^{(l)} \circ \mathbf{a}^{(l)}\right)^T \frac{\partial Q}{\partial \mathbf{z}_b^{(l)}} \\ \frac{\partial Q}{\partial \mathbf{c}^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}}\frac{\partial \mathbf{z}_r^{(l)}}{\partial \mathbf{c}^{(l)}} = \sum \frac{\partial Q}{\partial \mathbf{z}_b^{(l)}}\end{aligned} \tag{13}$$

and

$$\begin{aligned}\frac{\partial Q}{\partial \mathbf{a}^{(l)}} &= \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}}\frac{\partial \mathbf{z}_r^{(l)}}{\partial \mathbf{a}^{(l)}} + \frac{\partial Q}{\partial \mathbf{z}_g^{(l)}}\frac{\partial \mathbf{z}_g^{(l)}}{\partial \mathbf{a}^{(l)}} + \frac{\partial Q}{\partial \mathbf{z}_b^{(l)}}\frac{\partial \mathbf{z}_b^{(l)}}{\partial \mathbf{a}^{(l)}} \\ &= \frac{\partial Q}{\partial \mathbf{z}_r^{(l)}}\left(\mathbf{w}_r^{(l)}\right)^T + \frac{\partial Q}{\partial \mathbf{z}_g^{(l)}}\left(\mathbf{w}_g^{(l)}\right)^T \\ &\quad + 2\mathbf{a}^{(l)}\left[\frac{\partial Q}{\partial \mathbf{z}_b^{(l)}}\left(\mathbf{w}_b^{(l)}\right)^T\right]\end{aligned} \tag{14}$$

In contrast to the forward computation, the input to the BP procedure is the predicted result $\mathbf{y}$, which is the output of the forward process. For layer $l = L, \ldots, 1$, one layer at a time, we compute $\partial Q/\partial \mathbf{z}^{(l)}$ using Eqs. 9 or 10 depending on whether it is the last layer, the same for the conventional and quadratic neural networks. Then, we compute $\partial Q/\partial \theta^{(l)}$ according to Eqs. 11 (for conventional neurons) or 13 (for quadratic neurons) respectively, where θ denotes a vector of all trainable parameters of the network. Finally, we compute $\partial Q/\partial \mathbf{a}^{(l)}$, which is used in Eqs. 12 (for conventional neurons) and 14 (for quadratic neurons) respectively for the next iteration. After the gradient of the network is obtained, we update the parameters via 'Adam' in this study.

Whole training process

Initiation. Let us use a series of integers to describe a feed-forward neural network architecture of our interest,

$$\mathbf{d} = d^{(1)}, \cdots, d^{(l)}, \cdots, d^{(L+1)} \tag{15}$$

where $d^{(l)}$ represents the dimension of $\mathbf{a}^{(l)}$. Then, the total number of neurons used in the network is $\sum_{l=1}^{L} d^{(l+1)}$. Note that $d^{(1)} = D$, the dimension of input samples, and $d^{(L+1)} = K$, the number of classes.

Then, the network can be randomly initialized with a vector of parameters $\theta^{(l)}$ for each layer. Specifically, for each layer $l \in [1, L]$, let d_from be the input dimension $d^{(l)}$ and d_to the output dimension $d^{(l+1)}$ Setting all weights – $\mathbf{w}^{(l)}$ for a conventional neural network and $\mathbf{w}_r^{(l)}, \mathbf{w}_g^{(l)}, \mathbf{w}_b^{(l)}$ for a quadratic neural network – and biases – $\mathbf{b}^{(l)}$ for a conventional network and $\mathbf{b}_r^{(l)}, \mathbf{b}_g^{(l)}, \mathbf{c}^{(l)}$ for a quadratic network – as follows:

$$\begin{aligned}\text{weight} &= 0.01 \cdot \texttt{np.random.randn(d\_from, d\_to)} \\ \text{bias} &= \texttt{np.zeros([1, d\_to])}\end{aligned}$$

where np stands for NumPy (version 1.23.0), a Python package. That is, the bias is a 1 by $d^{(l+1)}$ zero matrix, and the weight is a $d^{(l)}$ by $d^{(l+1)}$ matrix.

Optimization. As shown in Fig. 1, given a neural network we just initialized and a training dataset containing samples $\mathbf{x}$ including the corresponding labels $\mathbf{t}$, we

can repeat the forward computation and BP processes described in the above two subsections until the stopping criteria are satisfied. The cross-entropy losses on the training and validation samples will be estimated during the training process.

Results and discussion

Using the training methods in the preceding section, we optimized conventional and quadratic neural networks to solve a number of GMM-based classification problems. At the beginning, we solved a three-class problem in the two-dimensional (2D) space to illustrate the working principle. Then, we performed a systematic comparison between conventional and quadratic neural networks on samples with different numbers of classes and dimensions. Finally, we applied all methods on three real data sets. Meanwhile, we used the EM algorithm and Bayes inference to obtain the Bayes error rate as the performance benchmark of the neural networks.

Illustrative classification example

Our initial classification problem assumes a finite number of classes (the first example, $K = 3$) in the 2D space ($D = 2$): two Gaussian clusters plus a background, which can be viewed as a special case of the Gaussian distribution. As in other network-based classification networks, a one-hot vector was used in our networks as well. The parameters of the background were set to

$$\mu_b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{C}_b = \begin{bmatrix} 40 & 0 \\ 0 & 40 \end{bmatrix}$$

where b indicates the background. Then, we randomly set the parameters of the other Gaussian clusters as

$$\begin{aligned} \mu_k &= (\texttt{np.random.random(D)} - 0.5) * 10 \\ a &= \texttt{np.random.random((D, D))} * 2 - 1 \\ \mathbf{C}_k &= 2aa^T \end{aligned}$$

for $k = 1, \ldots, K-1$ where aa^T stands for matrix multiplication and `np` stands for NumPy (version 1.23.0), a Python package. Given the mean μ_k and covariance $\mathbf{C}_k$, we generated N_k points for each class except the background where $N_k \in [20000, 30000]$ was chosen randomly. Then, we generated $N_b = \sum_{k=1}^{K-1} N_k$ points for the background. The entire dataset was shuffled and split into the three parts: 50% as training samples, 20% as validation samples, and 30% as test samples. Figure 5 shows the scatter plot of sample points.

We trained conventional and quadratic neural networks with different numbers of neurons for GMM classification. The decreasing loss is shown in Fig. 6 on the validation samples during the training process. The decision boundaries are shown in Fig. 7 for the conventional and quadratic neural networks as well as EM algorithm respectively. It took hundreds of neurons for the conventional network to approach the elliptical boundaries, while the quadratic network accurately fitted them with only three quadratic neurons.

The lighter the network structure, the higher the computation efficiency. Table 1 shows time spent to train the conventional and quadratic neural networks, and the accuracies of the EM and neural networks on the test samples. Our quadratic neural network with only one neural layer produced a performance closer to the EM benchmark than the conventional neural network of more than one hundred conventional

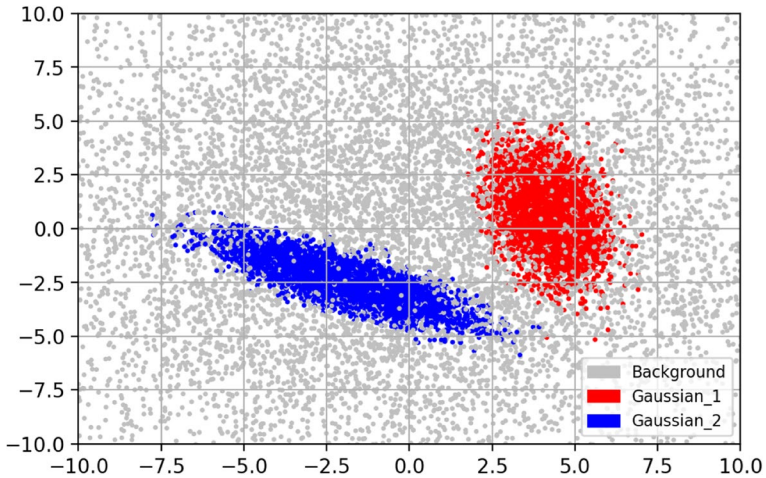

Fig. 5 Scatter plot of sample points which contains three classes: two Gaussian clusters plus a background

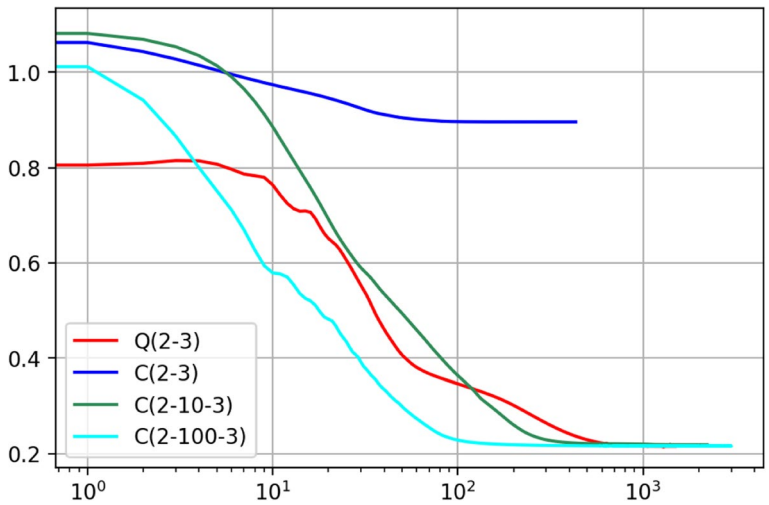

Fig. 6 Decreasing loss on the validation samples during the training process of conventional (C) and quadratic (Q) neural networks with different numbers of neurons. The notation $(\cdot)$ stands for the architecture of a neural network as described in Eq. 15

neurons. Also, the time need for the quadratic neural network is only about 7% that of the conventional counterpart.

Systematic comparation

To systematically compare conventional and quadratic networks, we tested conventional and quadratic networks in 2D and three-dimensional (3D) spaces with $K = 5$ and 8 Gaussian clusters. In each case, we randomly generated 50 samples using the aforementioned method except we replaced the background by a Gaussian cluster and set $N_k \in [6000, 9000]$. Typical scatter plots of these samples are represented in Fig. 8.

We trained and tested the EM algorithm, conventional and quadratic networks with different numbers of layers/neurons in terms of the average accuracy. The resutls are summarized in Table 2. Very interestingly, in all cases, the accuracy of the quadratic networks with only output layer of few neurons is higher than that of the conventional network of over one hundred neurons. Meantime, the training time needed for quadratic neural networks is only about 26.82%, on average, of that taken by the much more complicated conventional network. Generally speaking, the quadratic neural networks delivered a performance very close to that of the EM algorithm.

Real data

Finally, we applied conventional and quadratic networks on three real data sets from the UCI Machine Learning repository [20]: protein localization sites (yeast), pen-based recognition of handwritten digits (pendigits), and isolated letter speech recognition (isolet). All three data sets' attribute types are numerical. Some basic information about these datasets are in Table 3. For the yeast dataset, we split the whole dataset in the same proportions as that described in the first subection. Typical yeast cell (Saccharomyces cerevisiae cell) images the Cell Image Library [21] are shown in Fig. 9, visualized through transmission electron microscopy. For the pendigits and isolet datasets, with the test samples already provided, 30% of training samples were used for validation.

We trained and tested the EM algorithm, conventional and quadratic networks with different numbers of layers/neurons on each dataset 20 times. The average accuracy of and time needed by each method are shown in Table 4. In each application, the quadratic neural network with only layer of few neurons has the highest accuracy while its training time is about half of the conventional networks orders of magnitude larger than the quadratic version.

Conclusions

Although it has been well tested with a solid theoretical foundation, the EM algorithm needs to take an entire dataset into the memory, processes them iteratively, and is time-consuming, under the restriction that data must come from GMM. Furthermore, when new samples become available, parameters need to be adjusted again. A neural network approach can be much more desirable, effective and efficient, workable with many data models in principle thanks to its universal approximation nature. After a network is well trained, new

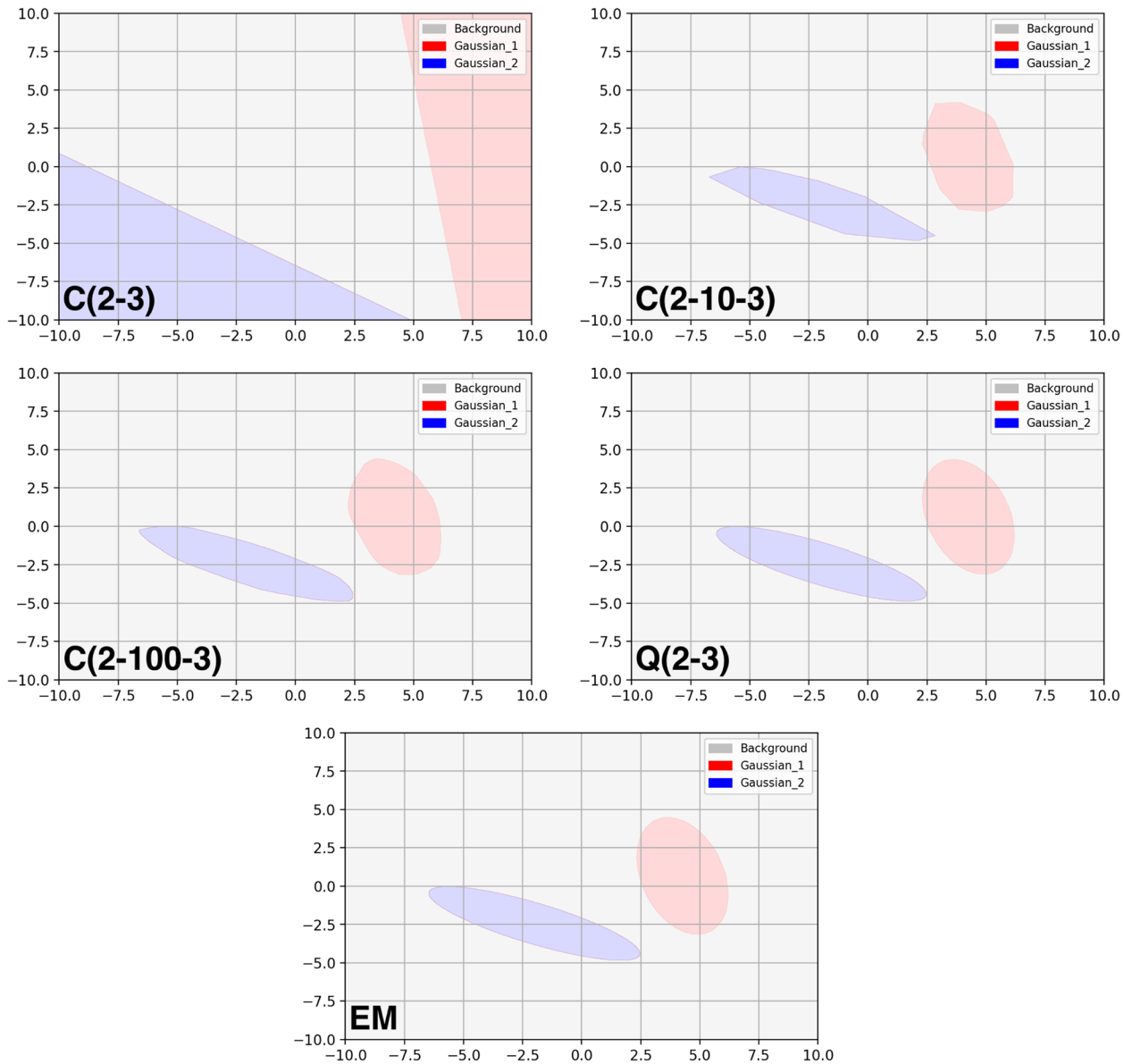

Fig. 7 Decision boundaries of EM and neural networks, including C and Q neural networks with different numbers of neurons. The notation $(\cdot)$ stands for the architecture of a neural network as described in Eq. 15

Table 1 The average accuracy of and time needed by EM algorithm, Q and C neural networks with different numbers of neurons in 2D spaces with two Gaussian clusters plus a background. The notation $(\cdot)$ stands for the architecture of a neural network as described in Eq. 15

	Accuracy (%)	Time (s)
C(2-3)	31.1752	2.44
C(2-10-3)	91.8122	23.15
C(2-100-3)	91.8391	213.61
Q(2-3)	91.8525	16.30
EM	91.8625	

samples can be used to fine-tune the network or processed to inference in a feed-forward fashion, being extremely efficient and generalizable to cases much more complicated than GMM. Very interestingly, compared to conventional networks, quadratic networks can deliver a performance close to that of the EM algorithm in the GMM cases and yet be orders of magnitude simpler than conventional networks for the same classification task.

In conclusion, in this paper we have numerically and experimentally demonstrated the superiority of quadratic networks over conventional ones. It is underlined

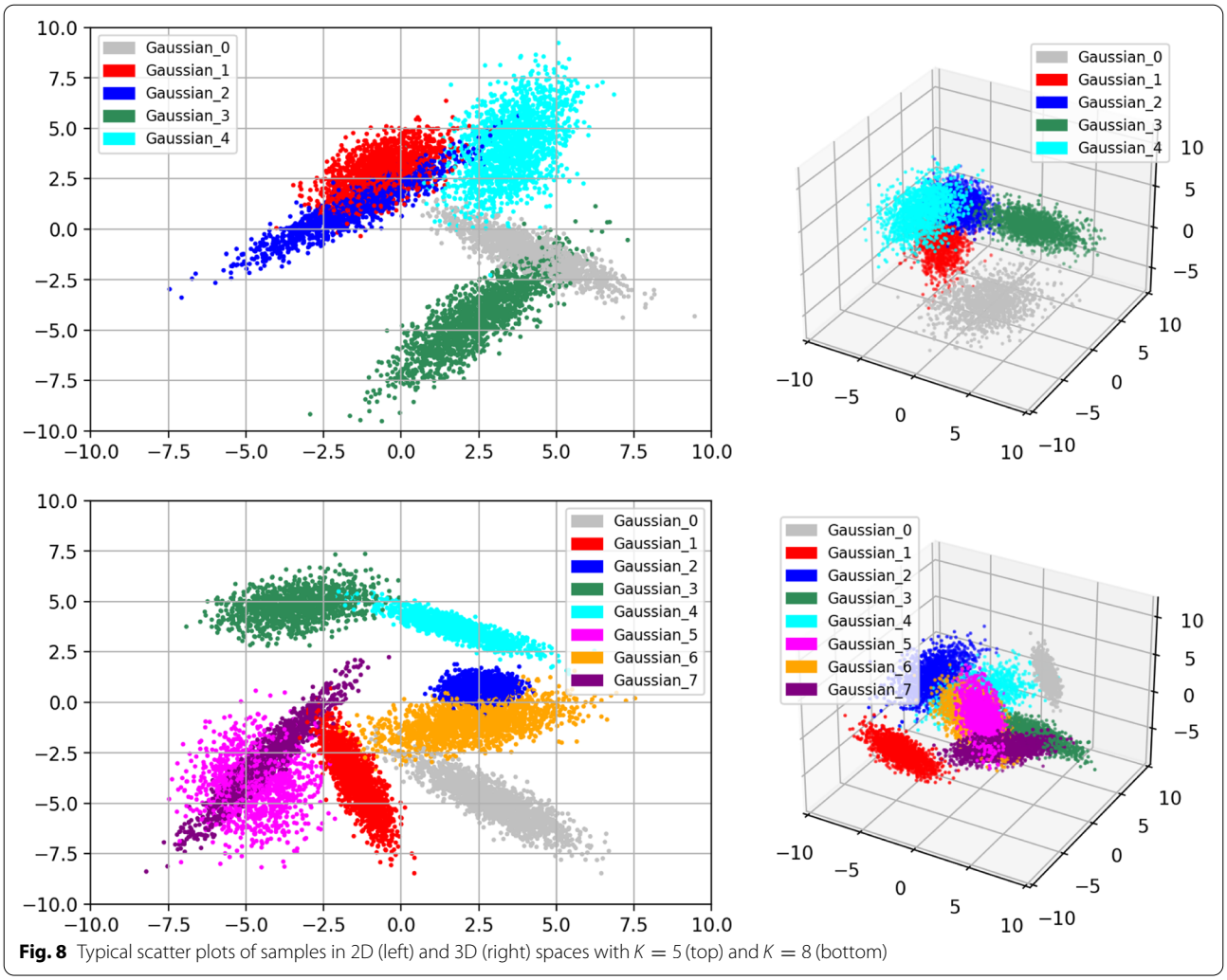

Fig. 8 Typical scatter plots of samples in 2D (left) and 3D (right) spaces with $K = 5$ (top) and $K = 8$ (bottom)

Table 2 The average accuracy of and time needed by EM algorithm, Q and C neural networks with different numbers of neurons in 2D and 3D spaces with $K = 5$ and 8 Gaussian clusters. The notation $(\cdot)$ stands for the architecture of a neural network as described in Eq. 15

	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)
	D = 2, K = 5		D = 3, K = 5	
C(2-3)	92.36 ± 7.89	11.5 ± 5.0	84.72 ± 10.32	13.7 ± 5.8
C(2-10-3)	95.36 ± 5.66	23.3 ± 14.1	92.15 ± 8.04	24.8 ± 8.6
C(2-100-3)	95.47 ± 5.68	62.7 ± 27.9	92.54 ± 8.01	71.8 ± 29.2
Q(2-3)	95.53 ± 5.66	15.8 ± 6.7	92.74 ± 7.98	17.6 ± 7.4
EM	95.60 ± 5.65		92.90 ± 7.97	
	D = 2, K = 8		D = 3, K = 8	
C(2-3)	83.84 ± 5.53	22.8 ± 8.0	76.47 ± 6.31	28.8 ± 10.6
C(2-10-3)	87.90 ± 3.75	47.6 ± 17.1	82.45 ± 5.36	62.6 ± 18.5
C(2-100-3)	88.06 ± 3.70	122.6 ± 39.2	82.68 ± 5.34	151.8 ± 41.4
Q(2-3)	88.13 ± 3.67	33.3 ± 11.0	82.79 ± 5.31	46.2 ± 13.3
EM	88.19 ± 3.64		82.86 ± 5.32	

Table 3 Basic information about three real data sets: protein localization sites (yeast), pen-based recognition of handwritten digits (pendigits), and isolated letter speech recognition (isolet)

Datasets	Train	Test	Dimensions	Classes
yeast	1484		8	10
pendigits	7494	3498	16	10
isolet	6238	1559	617	26

that the quadratic neural network of a much lighter structure rivals the conventional network of a complexity orders of magnitude more in solving the same classification problems. Clearly, the superior classification performance of quadratic networks could be translated to medical imaging tasks, especially radiomics.

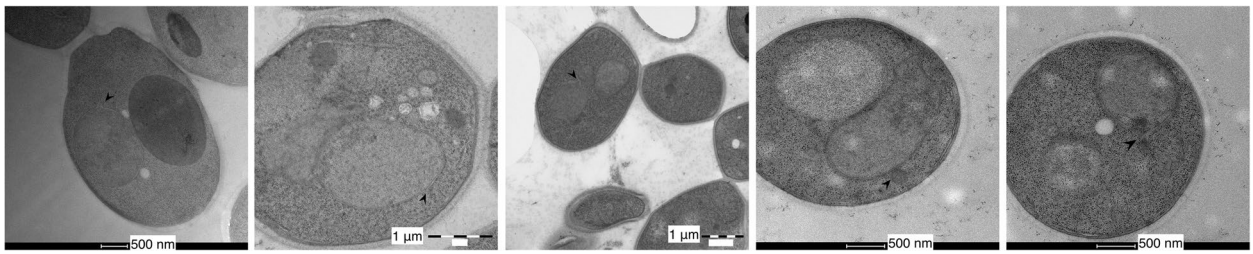

Fig. 9 Typical yeast images from the Cell Image Library (http://cellimagelibrary.org/groups/50815)

Table 4 The average accuracy of and time needed by Q and C neural networks with different numbers of neurons for three real datasets. The notation $(\cdot)$ stands for the architecture of a neural network as described in Eq. 15

	Yeast		Pendigits		Isolet	
	Accuracy (%)	**Time (s)**	**Accuracy (%)**	**Time (s)**	**Accuracy (%)**	**Time (s)**
C(2-3)	57.17 ± 1.79	0.5 ± 0.3	90.01 ± 1.54	3.3 ± 2.7	89.94 ± 3.30	11.2 ± 0.2
C(2-10-3)	58.21 ± 1.93	0.7 ± 0.2	93.23 ± 2.39	4.0 ± 1.9	92.00 ± 0.60	11.8 ± 0.7
C(2-100-3)	59.69 ± 2.96	0.9 ± 0.2	96.66 ± 0.29	14.8 ± 27.4	94.47 ± 0.30	49.8 ± 2.6
Q(2-3)	60.99 ± 1.27	0.5 ± 0.1	97.04 ± 0.30	6.1 ± 0.2	95.01 ± 0.17	21.2 ± 0.2

Abbreviations
ANN: Artificial neural network; BP: Backpropagation; EM: Expectation-maximization; GMM: Gaussian mixture model; 2D: Two-dimensional; 3D: Three-dimensional.

Acknowledgements
Not applicable.

Authors' contributions
GW suggested this research topic. TRQ designed the networks and experiments, and drafted the paper. Both discussed data analysis and revised the paper. All authors read and approved the final manuscript.

Funding
This work was supported in part by NIH, Nos. R01CA237267, R01HL151561, R21CA264772, and R01EB032716.

Availability of data and materials
The datasets analysed during the current study are available in the UCI Machine Learning repository, http://archive.ics.uci.edu [20]. Applications and source codes are available at https://github.com/tianrui-qi/QuadraticNeurons.

Declarations

Consent for publication
All authors give consent for publication.

Competing interests
The authors declare that they have no competing interests.

Received: 6 June 2022 Accepted: 16 August 2022
Published online: 28 September 2022

References

1. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, et al (2020) Language models are few-shot learners. Adv Neural Informat Proc Syst **33**:1877-1901
2. Sakaguchi, K., Le Bras, R., Bhagavatula, C., Choi, Y.: Winogrande: An adversarial winograd schema challenge at scale. Proceedings of the AAAI Conference on Artificial Intelligence **34**(05), 8732-8740 (2020)
3. Di Biase, G., Blum, H., Siegwart, R., Cadena, C.: Pixel-wise anomaly detection in complex driving scenes. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 16918-16927 (2021)
4. Liu, Y., Zhang, J., Fang, L., Jiang, Q., Zhou, B.: Multimodal motion prediction with stacked transformers. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 7577-7586 (2021)
5. Ma, X., Zhang, Y., Xu, D., Zhou, D., Yi, S., Li, H., et al.: Delving into localization errors for monocular 3d object detection. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 4721-4730 (2021)
6. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al (2019) Grandmaster level in starcraft II using multi-agent reinforcement learning. Nature **575**(7782):350-354. https://doi.org/10.1038/s41586-019-1724-z.
7. Moen E, Bannon D, Kudo T, Graf W, Covert M, Van Valen D (2019) Deep learning for cellular image analysis. Nat Methods **16**(12):1233-1246. https://doi.org/10.1038/s41592-019-0403-1.
8. Isensee F, Jaeger PF, Kohl SAA, Petersen J, Maier-Hein KH (2021) nnU-net: a self-configuring method for deep learning-based biomedical image segmentation. Nat Methods **18**(2):203-211. https://doi.org/10.1038/s41592-020-01008-z.
9. Wang G, Ye JC, De Man B (2020) Deep learning for tomographic image reconstruction. Nat Mach Intell **2**(12):737-748. https://doi.org/10.1038/s42256-020-00273-z.
10. Bennett KP, Brown EM, De Los Santos H, Poegel M, Kiehl TR, Patton EW, et al (2019) Identifying windows of susceptibility by temporal gene analysis. Sci Rep **9**(1):2740. https://doi.org/10.1038/s41598-019-39318-8.
11. Petegrosso R, Li ZL, Kuang R (2020) Machine learning and statistical methods for clustering single-cell RNA-sequencing data. Brief Bioinform **21**(4):1209-1223. https://doi.org/10.1093/bib/bbz063.
12. Arunkumar N, Mohammed MA, Ghani MKA, Ibrahim DA, Abdulhay E, Ramirez-Gonzalez G, et al (2019) K-means clustering and neural network for object detecting and identifying abnormality of brain tumor. Soft Comput **23**(19):9083-9096. https://doi.org/10.1007/s00500-018-3618-7.
13. Huang H, Meng FZ, Zhou SH, Jiang F, Manogaran G (2019) Brain image segmentation based on FCM clustering algorithm and rough set. IEEE Access 7:12386-12396. https://doi.org/10.1109/ACCESS.2019.2893063.

14. Fan FL, Cong WX, Wang G (2018) A new type of neurons for machine learning. Int J Numer Methods Biomed Eng **34**(2):e2920. https://doi.org/10.1002/cnm.2920.
15. Fan FL, Cong WX, Wang G (2018) Generalized backpropagation algorithm for training second-order neural networks. Int J Numer Methods Biomed Eng **34**(5):e2956. https://doi.org/10.1002/cnm.2956.
16. Fan FL, Shan HM, Kalra MK, Singh R, Qian GH, Getzin M, et al (2019) Quadratic autoencoder (Q-AE) for low-dose CT denoising. IEEE Trans Med Imaging **39**(6):2035-2050. https://doi.org/10.1109/TMI.2019.2963248.
17. Fan, F., Shan, H., Gjesteby, L., Wang, G.: Quadratic neural networks for CT metal artifact reduction. Developments in X-Ray Tomography XII **11113**, 111130 (2019). International Society for Optics and Photonics.
18. Fan FL, Wang G (2020) Fuzzy logic interpretation of quadratic networks. Neurocomputing 374:10-21. https://doi.org/10.1016/j.neucom.2019.09.001.
19. Fan FL, Xiong JJ, Wang G (2020) Universal approximation with quadratic deep networks. Neural Netw **124**:383-392. https://doi.org/10.1016/j.neunet.2020.01.007.
20. Dua, D., Graff, C.: UCI Machine Learning Repository (2017). http://archive.ics.uci.edu/ml. Accessed 2022-05-28.
21. Höög, J., Panagaki, D., Croft, J.: CIL:50813 - 50817, Saccharomyces cerevisiae (baker's yeast, budding yeast), Mixed population of S. cerevisiae cells. CIL. Dataset. (2020).http://cellimagelibrary.org/groups/50815. Accessed 2022-05-28.

Publisher's Note