Looping back: Circular nodes revisited with novel applications in the radio frequency domain

Tim Marrinan
Pacific Northwest National Laboratory
timothy.marrinan@pnnl.gov

Bill Kay
Pacific Northwest National Laboratory
william.kay@pnnl.gov

Audun Myers
Pacific Northwest National Laboratory
audun.myers@pnnl.gov

Rachel Wofford
Pacific Northwest National Laboratory
re.wofford@pnnl.gov

Tegan Emerson
Pacific Northwest National Laboratory
University of Texas at El Paso
tegan.emerson@pnnl.gov

Abstract

In domains with complex-structured data, some relationships cannot be easily modeled using only real-valued Euclidean features. In spite of this misalignment, most modern machine learning methods default to representing data in just that way. By failing to appropriately encode the data structure, the performance and reliability of the resulting machine learning models can be degraded. In prior work, Kirby and Miranda introduced the concept of a circular node, a type of artificial neuron engineered to represent periodic data or angular information [11]. These nodes can be implemented directly in many traditional neural network architectures to more faithfully model periodic relationships. However, since they have garnered relatively little attention compared to their non-circular counterparts, circular nodes have largely been excluded from open-source machine learning libraries. In this paper, we re-investigate circular nodes in the context of modern machine learning libraries, and demonstrate the advantages of circular representations in applications with complex-structured data. Our experiments center around radio frequency signals, which naturally encode circular relationships. We illustrate that a neural network composed of a single circular node can learn the phase offset of a radio frequency signal. We show that a fully-connected neural network made up of multiple layers of circular nodes can successfully classify digital modulation constellation points, and demonstrates accuracy gains over its traditional counterpart when the model size is small. Finally, we demonstrate notable performance improvements on the task of automatic modulation classification through the integration of a circular node layer into traditional convolutional networks.

1 Introduction

Artificial Intelligence (AI) hinges on data that mathematically represents content, function, or observations. The success of AI has led to the development of mature and accessible tools that allow anyone to create neural networks of their own. However, leveraging open-source machine learning libraries typically requires representing problems using real-valued Euclidean features. While these representations are natural for RGB images or quantified variable properties, other domains

1st Conference on Topology, Algebra, and Geometry in Data Science (TAG-DS 2025).

present challenges. In natural language processing, for example, vector space representations are not straightforward and require sophisticated tools to find appropriate features [5].

Some problems are fundamentally ill-suited to Euclidean representations because they possess inherent mathematical structure that must be preserved. Periodic data appears in many fields including physics, communications, atmospheric science, and biology. Consider circadian rhythms, largely controlled by clock genes, which significantly impact human health and behavior [1, 2, 3, 15]. Disruptions to these genetic rhythms are associated with pathologies including Type-II diabetes, obesity, and Alzheimer's disease [13, 14, 20, 21]. In communications, phase-shift keying (PSK) is used to transmit information by modulating the phase of a carrier signal, requiring accurate phase detection to decode the transmitted messages [17]. In atmospheric and oceanic modeling, it has been shown that traditional models are too simplistic to describe phenomena like the El Niño-Southern Oscillation and the stratospheric quasi-biennial oscillation [8]. In each of these examples, if we force the data with naturally occurring periodicity into a real-valued vector space representation, we throw away any notion of the phase that might have been present.

Ignoring intrinsic mathematical structure and relationships can come at the expense of interpretability, reliability, and robustness. For these reasons there is still a need for machine learning architectures that understand, preserve, or leverage complex mathematical structure and relationships. However, mathematically-inspired neural designs often require more computational resources than their generic counterparts. In this manuscript we interrogate a neural network architecture designed to faithfully represent periodic relationships, assessing the value of encoding these relationships explicitly versus relying on traditional networks to learn these patterns.

In traditional neural networks, neurons encode state values on an interval of the real line, $\mathbb{R}^1$. In contrast, the so-called *circular node* consists of a coupled pair of nodes whose activations are constrained to live on the circle, S^1 [11]. As a result, the state values are also constrained to a real-valued 1-dimensional manifold, but they allow for a more natural representation of periodicity. This unique node-type can be integrated into conventional components like feed-forward layers or convolutional layers. In this manuscript, we demonstrate the practical utility of circular nodes using communications examples, and identify scenarios where missing phase information causes traditional models to fail. We focus on Radio Frequency (RF) signals because the in-phase (I) and quadrature (Q) components naturally exhibit circular relationships.

Another relevant type of mathematically-inspired neural design is the complex-valued neural network (CVNN) [7, 9, 10]. In CVNNs, the inputs, weights, biases, and activations can all take on values in an interval of $\mathbb{C}^1$. In the RF tasks that follow, CVNNs might offer performance improvements over real-valued networks, however our goal is not to establish circular nodes as optimal, but rather to demonstrate that they provide computational advantages when augmenting traditional architectures. Additionally, complex-backpropagation can be implemented using the Wirtinger derivative, which allows for the inclusion of activation functions that are not holomorphic [4, 6, 22]. As a result, a *complex circular node* could be introduced for CVNNs in future work. However, since the computational libraries for CVNNs are not as mature as those for real-valued networks, such an extension and comparison is beyond the scope of this paper.

The main contribution of this manuscript is to evaluate the efficacy of circular nodes as a method for encoding angular information in neural networks. To this end, we review the design and implementation of the circular nodes in Section 2. We consider three experiments in digital communications using RF data. As a proof-of-concept, in 3, we show that a simple neural network with a circular bottleneck layer can learn the angle of (noisy) data sampled from the unit circle. In 4, we extend this demonstration to show that a fully-connected neural network with layers composed of circular nodes can be used for effectively classifying digital modulation constellation points, especially in low-capacity networks. In 5, we show that using a layer of circular nodes to preprocess signals can significantly improve the results of automatic modulation classification as compared to the traditional approach of using a convolutional network alone. Throughout the experiments we return to a common refrain, using circular nodes to represent angular or periodic data offers representational benefits when the networks have low capacity. Finally, Section 6 discusses future directions for mathematically inspired neural design.

2 Background

Consider a neural network with L layers, where the input layer is labeled 0 and the output layer is labeled $L-1$. Suppose that layer i has $N^{(i)}$ nodes, and the jth node is indexed as $\mathcal{N}_j^{(i)}$ for $j = 0, \ldots, N^{(i)} - 1$. At node $\mathcal{N}_j^{(i)}$ the state value is denoted $\mathcal{S}_j^{(i)}$. Following the formulation of Kirby and Miranda, a *circular node* is an abstract node represented by a coupled pair of nodes, $\left\{\mathcal{N}_j^{(i)}, \mathcal{N}_{\tau(j)}^{(i)}\right\}$, that satisfy the circular constraint,

$$(\mathcal{S}_j^{(i)})^2 + (\mathcal{S}_{\tau(j)}^{(i)})^2 = 1, \tag{1}$$

by design, where the index mapping τ is defined to be an involution so that $\tau(\tau(j)) = j$ [11]. Thus n circular nodes will contain $2n$ state values that represent n angles. For each node within a coupled pair, the state value is computed as

$$\mathcal{S}_j^{(i)} = \frac{\mathcal{P}_j^{(i)}}{\sqrt{(\mathcal{P}_j^{(i)})^2 + (\mathcal{P}_{\tau(j)}^{(i)})^2}}, \tag{2}$$

where $\mathcal{P}_j^{(i)}$ is a prestate value defined using the output of the preceding layers as

$$\mathcal{P}_j^{(i)} = \sigma_j^{(i)} \left(b_j^{(i)} + \sum_{k=0}^{N^{(i-1)}-1} w_{kj}^{(i-1)} \mathcal{S}_k^{(i-1)} \right), \tag{3}$$

and $b_j^{(i)}$, $w_{kj}^{(i-1)}$, and $\sigma_j^{(i)}$ represent the biases, weights, and activation functions of the associated layers and nodes. The work of Kirby and Miranda goes on to derive the back-propagation algorithm for a fully-connected network consisting of circular nodes and sigmoidal nodes using the total squared error as the loss function, however with modern open-source software, these gradient updates can be handled automatically [11].

Circular nodes have found practical applications describing cyclic patterns and nonlinear phenomena. In particular, the nonlinear generalization of principal component analysis (nonlinear PCA) [12] can accommodate a circular node bottleneck layer. This strategy has been used to model atmospheric and oceanic phenomena [8], and was implemented as part of a nonlinear PCA Matlab toolbox that was used to describe the transcriptional variation of the malaria parasite *Plasmodium falciparum* throughout the intraerythrocytic developmental cycle [18, 19]. Following a similar paradigm, algorithms with a circular bottleneck layer have been developed for studying the circadian rhythms of gene expression in a variety of other applications [1, 2, 3, 15]. The commonality between these methods is that they rely on a single circular node to extract phase information as part of the bottleneck layer in an autoencoder architecture. Expanding on these approaches, we investigate how circular node layers can be more fully integrated into existing neural network architectures by either replacing or supporting traditional nodes.

2.1 Circular data in communications

Communications applications are often concerned with encoding and transmitting a source signal, $x(t)$, by modulating a carrier signal, $c(t)$, that itself contains no information [17]. A cosine wave, $c(t) = A\cos(2\pi f_c t)$, is typically used as the carrier signal, where A is the amplitude and f_c is the carrier frequency. In digital communications specifically, this source signal $x(t)$ is digitized to produce a discrete signal, $u(t)$, by sampling, quantization, and coding. We can think of these baseband symbols as weighted impulses to which we have applied a pulse shape, such as a rectangular pulse or a raised cosine pulse.

Digital modulation types include amplitude-shift keying (ASK), frequency-shift keying (FSK), phase-shift keying (PSK), pulse amplitude modulation (PAM), and quadrature amplitude modulation (QAM), among others. A digitally modulated signal is constructed as

$$s(t) = A(t)\cos(2\pi f_c t + \theta(t)), \tag{4}$$

where $A(t)$ is the time-varying amplitude and $\theta(t)$ represents the phase-shift.

As an example that introduces the data with which the experiments are conducted, let us consider PSK data. For PSK, $A(t)$ is held constant and the phase-shift can take on a range of values depending on the modulation scheme. For example, with binary phase-shift keying (BPSK), $\theta(t) \in \{0, \pi\}$ and for quadrature phase-shift keying (QPSK) the discrete signal can take four different values, $\theta(t) \in \{-\frac{3\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{4}, \frac{3\pi}{4}\}$. Using the identity $\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$, we can write the QPSK signal in terms of an amplitude modulated quadrature carrier, where we have

$$s(t) = \begin{cases} -\frac{\sqrt{2E_s}}{2}\cos(2\pi f_c t) + \frac{\sqrt{2E_s}}{2}\sin(2\pi f_c t), & \text{if } \theta(t) = -\frac{3\pi}{4} \\ +\frac{\sqrt{2E_s}}{2}\cos(2\pi f_c t) + \frac{\sqrt{2E_s}}{2}\sin(2\pi f_c t), & \text{if } \theta(t) = -\frac{\pi}{4} \\ +\frac{\sqrt{2E_s}}{2}\cos(2\pi f_c t) - \frac{\sqrt{2E_s}}{2}\sin(2\pi f_c t), & \text{if } \theta(t) = \frac{\pi}{4} \\ -\frac{\sqrt{2E_s}}{2}\cos(2\pi f_c t) - \frac{\sqrt{2E_s}}{2}\sin(2\pi f_c t), & \text{if } \theta(t) = \frac{3\pi}{4}, \end{cases} \tag{5}$$

where $\sqrt{E_s}$ is the peak amplitude of the modulated carrier. This signal can then be represented by the momentary amplitude of its I and quadrature Q components, which demonstrates the relationship between PSK and ASK modulation. Considering the amplitude coefficients as weighted impulses that are used to modulate the carrier signal provides the I/Q data used for the following experiments. For descriptions of other modulation schemes, please refer to a standard text on digital communications, such as [17].

The most popular communication channel in signal processing is the additive white Gaussian noise (AWGN) channel which is characterized by constant spectral density and a Gaussian amplitude distribution of zero mean. In the experiments that follow, we consider AWGN data, and leave other channels like fading channels and non-Gaussian channels as future work.

2.2 Radio frequency use-case and implementation

RF signals provide an ideal use-case for circular nodes because they naturally consist of I and Q components that exhibit circular relationships. For our experiments, we adapted the RF signal generator described in [16] to produce synthetic noisy I and Q signals with configurable properties. Throughout our experiments, we specify the signal length, signal-to-noise ratio (SNR), and modulation types used for each evaluation.

While [11] establishes the fundamental operating mechanism of circular nodes, implementation details for practical applications require further development. We have extended their basic circular node structure to create a specialized implementation for handling I and Q signals in RF applications.

The circular node layer implementation consists of a coupled pair of linear transformations that map input features to a unit circle representation while preserving magnitude information. Given input signals, $\mathbf{I}, \mathbf{Q} \in \mathbb{R}^{d_{in}}$, the circular layer computes the prestate values

$$\begin{aligned} \mathbf{P}_N &= W_N \mathbf{I} + \mathbf{b}_N \\ \mathbf{P}_T &= W_T \mathbf{Q} + \mathbf{b}_T \end{aligned} \tag{6}$$

where $W_N, W_T \in \mathbb{R}^{d_{out} \times d_{in}}$ are learned weight matrices, and $\mathbf{b}_N, \mathbf{b}_T \in \mathbb{R}^{d_{out}}$ are bias vectors. The magnitude of the vector in the I/Q-plane is computed as $\mathbf{R} = \sqrt{\mathbf{P}_N^2 + \mathbf{P}_T^2}$, and the state values are $\mathbf{S}_N = \mathbf{S}_N / \max(\mathbf{R}, \epsilon)$ and $\mathbf{S}_T = \mathbf{S}_T / \max(\mathbf{R}, \epsilon)$, respectively, where $\epsilon = 10^{-16}$ prevents division by zero and the operations are applied elementwise. The pre-activation values $\mathbf{P}_N$ and $\mathbf{P}_T$ represent the raw linear transformations, while R captures the magnitude information. The final outputs $\mathbf{S}_N$ and $\mathbf{S}_T$ are constrained to lie on the unit circle, satisfying $\mathbf{S}_N^2 + \mathbf{S}_T^2 = \mathbf{1}$. Subsequent layers apply the same operations to the output of the preceding layers.

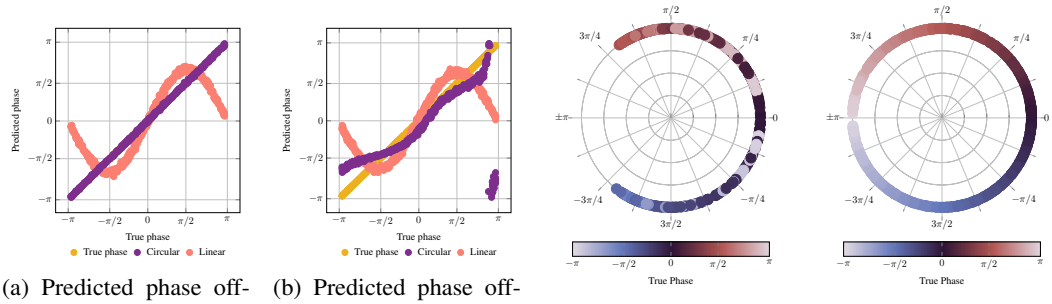
This formulation preserves angular information in the circular constraint and magnitude information in $\mathbf{R}$, allowing subsequent layers to utilize both geometric properties. The layer outputs, $(\mathbf{S}_N, \mathbf{S}_T, \mathbf{R})$, can be concatenated or processed separately depending on downstream requirements.

3 Phase-offset estimation

As a proof-of-concept, we conduct a comparative analysis between a simple single-layer linear neural network and a simple single-layer circular node neural network on the task of phase-offset estimation. Following the model in Equations (4), we assume knowledge of the carrier frequency, f_c,

Table 1: Phase-offset estimation fidelity

		Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	**Mean**	**Std.**
Circular	MAE	0.39	0.22	0.31	0.02	0.28	0.02	0.20	0.28	0.44	0.02	0.22	0.155
	RMSE	1.30	1.14	0.99	0.02	1.10	0.02	1.06	1.03	0.82	0.02	0.75	0.516
Linear	MAE	0.52	0.54	0.54	0.56	0.54	0.48	0.57	0.54	0.51	0.57	0.54	0.028
	RMSE	0.95	1.06	0.99	1.00	1.04	1.01	1.04	0.99	1.00	1.04	1.01	0.034



(a) Predicted phase offset for both of the models from Fold 10.

(b) Predicted phase offset for both of the models from Fold 1.

(c) Predicted phase offset for linear bottleneck.

(d) Predicted phase offset for circular bottleneck.

Figure 1: Comparison of predicted phase offsets using the linear bottleneck and the circular bottleneck. Results represent the predictions on the validation set from a single training run of each model.

and the (constant) amplitude, A, and attempt to estimate the phase offset, θ. We used complex-valued synthetically generated data, providing the ground truth phase offset for each signal and allowing us to evaluate model accuracy.

This experiment was conducted using synthetically generated data from the RF signal generator discussed in Section 2.2, modified to accommodate random phase offsets. The dataset is composed of 5000 complex-valued signals with 20 samples per signal. The signal duration is 2 seconds with a sample rate of 10 Hz, amplitude of 1.0 dB, and frequency of 1 Hz. Each signal is generated with random phase offset, θ, between $-\pi$ and π radians. We apply AWGN to the I and Q channels of each signal with a mean of 0 and a standard deviation of 0.1. The signals are split into training data subsets containing 4500 signals and validation data subsets containing 500 signals using k-fold cross-validation with $k = 10$.

To assess the base performance of the circular layer, we chose to conduct this experiment with very lightweight neural networks. We examine two distinct network architectures. For this experiment, the 'linear model' is composed of a single linear layer. The 'circular model' is composed of a single circular node bottleneck layer. The models are almost identical in size; the linear model consists of 41 parameters while the circular node model consists of 42 parameters. For the linear model, the predicted phase is the direct output of the bottleneck while the circular model outputs the element-wise inverse tangent of the bottleneck outputs.

Both networks receive the noisy I and Q data as input and output a single, continuous-valued phase prediction. The models use different methods to process the I and Q data. The linear model horizontally concatenates the I and Q vectors before passing the single concatenated vector through the linear layer. The circular model passes I and Q directly through the circular node layer then calculates the 4-quadrant inverse tangent of the circular node layer outputs. The models are trained for 20 epochs with a learning rate of 0.001 using the Adam optimizer and mean-squared error loss.

3.1 Results

The median absolute error (MAE) and root mean-squared error (RMSE) for each fold of the cross validation of the phase-offset estimation are reported in Table 1. In Figure 1, we plot the outputs of each network relative to the ground truth phase-offset for representative examples of each model type. The quantitative and qualitative results highlight some clear advantages of the circular node bottleneck. In Figure 1a we can see that the circular node network provides a much better fit to the true phase shifts than the linear network, which struggles to handle the branch cut at $-\pi/\pi$. In fact,

as we see in Table 1, the circular node network has an average MAE of 0.22 radians on the validation data across all folds, while the linear network has an average MAE of 0.54 radians. However, the standard deviation of the MAE for the circular models is 0.155, compared to 0.028 for the linear models, suggesting that not all folds are performing equally well. The high standard deviation of accuracy of the circular models is a result of predictions for noisy samples near the $-\pi/\pi$ branch cut being off by a full period. These predictions are measured as large mistakes, but in fact are very close to the true phase-offset modulo 2π. This mismatch between the measure of model fidelity and the periodicity of the data is present in the loss function used to train the models as well. In Figure 1b, we see one example of how a circular node network converges to a sub-optimal solution because the mean-squared error loss function misrepresents the error associated with points that represent equivalent phase offsets. The linear networks have a more difficult time addressing this challenge. Each of these networks converges to a solution where the phase-offsets for points near both sides of the branch cut are predicted as zero, so that nearby I/Q points will have similar predictions. Unfortunately this results in very large errors near the endpoints. Figure 1c and Figure 1d display the phase offsets predicted by the linear node network and the circular node network, respectively, for Fold 10 of the cross validation. The true value of the phase offsets is indicated via the color gradient and we can see that they are evenly distributed about the unit circle. We see a similar distribution for the predicted phases from the circular node network, with small gaps at a few intervals. Conversely, we see that the linear network only predicts phase offsets within the range $[-3\pi/4, 3\pi/4]$, which is consistent with the regression curve in Figure 1a.

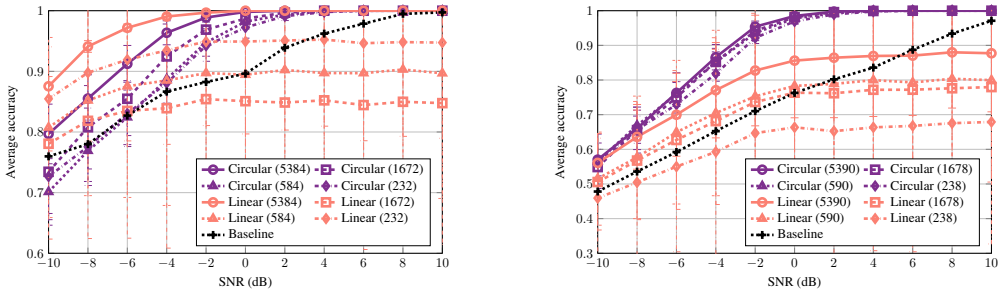
4 PSK demodulation

The phase-offset estimation experiment demonstrated the potential of the circular node layer to model the relationship between the real and imaginary components of complex-valued data. We now show that by effectively modeling this relationship, a network composed entirely of circular nodes can accurately identify constellation points in a PSK demodulation task, using BPSK and QPSK modulation schemes. The evaluation is conducted using the synthetically generated RF data introduced in section 2.2. The data contains the noisy I and Q channels of 2000 signals for SNRs ranging from -10 dB to 10 dB, as well as the ground truth symbol sequence for the signal, which is used for network training and evaluation. Each generated signal consists of 20 symbols with a symbol period of 2π and 8 samples per symbol. The signal blocks associated with each symbol are segmented, and the inputs into each network are the noisy I/Q data associated with individual symbols. We divide the signals evenly with a $50/50$ train-test split, which yields 1000 signals/20000 symbols per partition. The random splits are repeated for each of 10 independent trials. Each network is trained with only the 10 dB training partition, but is tested using the test partitions across all SNRs.

Two types of reference model are considered. The first type is a signal processing model, referred to as 'baseline', which uses the sum of the samples within a symbol period to estimate the symbol. For the BPSK constellation, the symbols are $\{1, 0\}$ which correspond to the angles $\{0, \pi\}$, respectively. Thus, the baseline model for BPSK demodulation only considers the in-phase signal, and predicts 0 when the sum of the in-phase samples is positive, and 1 when the sum is negative. For the QPSK constellation, the symbols are $\{00, 01, 10, 11\}$ which correspond to angles $\{-\frac{3\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{4}, \frac{3\pi}{4}\}$. In this case, the baseline model considers the sum of the in-phase samples and the quadrature samples separately according to whether the sums of I and Q samples are positive or negative.

The second type of reference model is a 3-hidden-layer fully-connected network with linear nodes and ReLU activations. We define three such networks of different sizes. These networks accept the I and Q channels as input and horizontally concatenate them before passing the concatenated vector through the networks. The layers of these networks have $\{8n, 4n, 2\}$ nodes per layer for $n = 8, 4, 2, 1$. The networks output the class corresponding to the predicted modulation symbol, with two classes possible for BPSK and four classes possible for QPSK. Thus, the resulting models have 5384, 1672, 584, and 232 parameters for BPSK classification and 5390, 1678, 590, and 238 parameters for QPSK classification.

To evaluate the potential benefits of the circular node layer, we define models with an equivalent number of parameters to those of the linear networks. This parity is achieved by including half as many circular nodes per layer, as there were linear nodes in the reference models. Thus, these circular networks have $\{4n, 2n, 1\}$ circular nodes per layer for $n = 8, 4, 2, 1$, and contain no additional activation functions. The bottleneck layer feeds directly into a classification head that outputs one



(a) Results of the BPSK demodulation experiment.

(b) Results of the QPSK demodulation experiment.

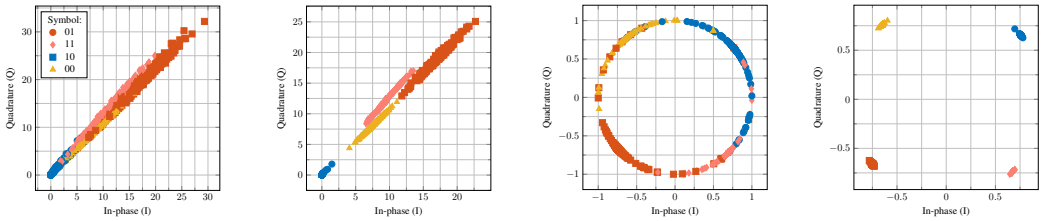
Figure 2: Average accuracy (across 10 trials) of predicting demodulated symbols from noisy I/Q data.

of two classes for BPSK or one of four classes for QPSK. As a result of the circular layers having half as many nodes as the linear layers, we again have 5384, 1672, 584, and 232 parameters for BPSK classification and 5390, 1678, 590, and 238 parameters for QPSK classification. We train each network for 25 epochs with a learning rate of 0.001 using the Adam optimizer and cross-entropy loss.

4.1 Results

In Figure 2 we see the average accuracy of all techniques for the two demodulation experiments, along with associated error bars, which represent one standard deviation from the mean. The BPSK and QPSK sub-tasks are essentially 2-class and 4-class classification problems. The baseline method shows us that the BPSK classification is a much easier task than the QPSK classification, and provides a standard for the accuracy that can be achieved with no training, but with knowledge of the geometry of the constellations points. For the fully-connected network with linear nodes, the input is simply an 8-dimensional vector, whose elements are noisy versions of the I and Q values that correspond to the BPSK and QPSK constellation points. With respect to the BSPK data, there is no ambiguity about the relationship between the constellation points, even for high levels of noise; one symbol is on the left and the other is on the right, so angular information is largely unnecessary. Thus, we expect this to be a straightforward task that for a traditional neural network to solve, and indeed we see from the results in Figure 2a that the networks can separate the two classes well across a wide range of SNRs. The largest linear network outperforms all others, while the smaller linear models fail to reach 100% average accuracy at high SNRs because the training converges to bad local minima for some of the training runs. For the model with the circular nodes, we see a different story. We use the term 'linear model' to represent the fully-connected network with linear nodes, but in fact that network is nonlinear and includes ReLU activation functions. The network with circular nodes, however, only contains affine transformations outside of the circular nodes, with no other nonlinear activation. In this context, it might be expected that greater network capacity is required to represent very noisy data. Thus it is not surprising that the circular models underperform their counterparts, including the baseline method, in the BPSK demodulation experiment at low SNRs, as we see in Figure 2a.

For the QPSK data, on the other hand, the constellation points are evenly distributed about the unit circle. When noise levels are high, the linear networks struggle to represent the circular relationship between the 4 classes, and the smallest model performs worse than the baseline method, while the circular models on average do much better than the comparison methods. Intuitively, this is because these networks are forced to represent the true underlying relationships between the constellation points. We can see this discrepancy between the linear and circular networks in the activations of their respective bottleneck layers, as shown in Figure 3 for the largest capacity networks of each type. In Figure 3a and Figure 3b, we see that the linear nodes try to map the QPSK constellation points to a line segment. In the high SNR case these clusters are still separable, but the relationships between the classes are not well preserved. For the circular models, on the other hand, we can see from the network activations in Figure 3c and Figure 3d that the class relationships are preserved at both noise levels and as the SNR increases, the clusters become easily separable. Our takeaway is that structurally encoding circular information offers better representations at lower network capacities, when angular information is needed. When the angles are not necessary for classification, using the circular nodes creates unnecessary computational burden, as we see in the BPSK results.



(a) Linear bottleneck activations with SNR 0 dB.

(b) Linear bottleneck activations with SNR 8 dB.

(c) Circular bottleneck activations with SNR 0 dB.

(d) Circular bottleneck activations with SNR 8 dB.

Figure 3: Visualization of 2-dimensional activations of the bottleneck layer for the fully-connected network with linear nodes and with circular nodes in the I/Q-plane for the QPSK demodulation task.

Table 2: Parameter counts for the circular and baseline architectures across model sizes.

	Circular + CNN	Linear + CNN	CNN
Model 1	3293	3298	3334
Model 2	877	882	902
Model 3	225	230	317
Model 4	98	103	99
Model 5	72	77	73

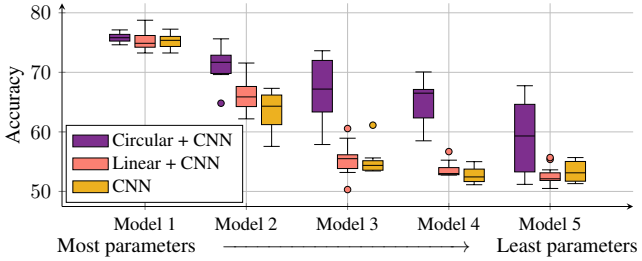


Figure 4: Automatic modulation classification accuracy as a function of network capacity.

5 Modulation classification

This experiment provides evidence that for certain tasks, traditional convolutional neural network (CNN) architectures benefit from an initial circular layer in resource-constrained settings such as edge devices. To this end, we designed three architectures: a traditional convolutional neural network (CNN), a comparable convolutional network with a circular layer between the input and convolutional layers (Circular + CNN), and the same base convolutional network with a linear layer between the input and convolution layers (Linear + CNN). We present five collections of models with decreasing parameter counts shown in Table 2 representing different parameter budgets. Each architecture includes hidden layers with configurable width parameters that can be scaled across different parameter budgets.

For the evaluation task, we adapted the RF signal generator described in [16] to produce noisy I and Q signals for modulation classification. We selected four PSK modulation schemes as target classes: BPSK, QPSK, 8PSK, and 16PSK. We focus on this task since preliminary experiments indicate circular nodes show particular effectiveness for phase information recognition. We fixed the SNR at 5 dB and generated 2000 I,Q pairs of length 128 per modulation type, using an 80/20 train-test split.

All architectures were trained using the Adam optimizer with a learning rate of 1e-3 and weight decay of 1e-6. Networks were initialized using Xavier uniform initialization and trained for 30 epochs with a batch size of 8 using cross-entropy loss. We report test accuracy as our primary evaluation metric, averaged over 10 independent runs for each parameter configuration.

5.1 Results

For each parameter configuration, we trained and tested all three architectures 10 times and report test accuracy metrics in Figure 4. Inclusion of the circular nodes shows value, though only marginal for the largest models, for all model sizes. As the parameter count decreases, the performance degrades for all architectures though notably slower for the configurations including circular nodes. The smallest instantiation shows a clear advantage for the circular layer approach. For each model, we conducted paired-samples t-tests comparing Circular+CNN accuracy against alternative architectures over 10 independent runs. The difference in performance is shown to be statistically significant across all model sizes with an upper bound of $(p < 0.05)$.

6 Conclusion and future directions

In this work we explored the utility of neural networks that encode the geometry of a problem domain by design. We showed that simple one-hidden-layer networks, fully-connected multi-layer networks, and convolutional neural networks can all benefit from the inclusion of circular nodes in RF tasks where the angular information, e.g., phase, of the signals contains information tightly coupled with downstream tasks. We find that in computationally constrained settings where the parameter budgets are restricted, the benefit of circular nodes is more significant. One potential criticism of circular node networks is that they require twice as many parameters as their generic counterparts for a network with the same number of nodes. We have addressed this issue a priori by comparing models with comparable parameter counts across all experiments.

This re-investigation of circular nodes has demonstrated their utility in a novel domain, where value is shown to be added across multiple tasks. However, there are cases where the performance variation of circular node networks is larger than desirable as a result of the mismatch between the choice of loss function and the periodic nature of the relationship being modeled. In future work, we intend to develop a family of loss functions that more fully exploit the structure encoded by the circular nodes. Moreover, there is open space to understand the best optimization approaches and learning objectives for hybrid architectures containing both circular nodes and others. Finally, with the ongoing improvements to open-source implementations of CVNN architectures, we intend to extend the circular node paradigm to networks with complex-valued parameters and data. We will be releasing a PyTorch compatible implementation of circular nodes to enable broader adoption and exploration within the community.

Acknowledgments and Disclosure of Funding

This research was supported by the Generative AI (GenAI) Investment, under the Laboratory Directed Research and Development (LDRD) Program at Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830.

References

[1] Ron C Anafi, Lauren J Francey, John B Hogenesch, and Junhyong Kim. CYCLOPS reveals human transcriptional rhythms in health and disease. *Proceedings of the National Academy of Sciences*, 114(20):5312–5317, 2017.

[2] Bharath Ananthasubramaniam and Ramji Venkataramanan. Rhythm profiling using COFE reveals multi-omic circadian rhythms in human cancers in vivo. *PLoS biology*, 23(5):e3003196, 2025.

[3] Aram Ansary Ogholbake and Qiang Cheng. PENN: Phase estimation neural network on gene expression data. In *The International Conference on Deep Learning, Big Data and Blockchain*, pages 59–67. Springer, 2023.

[4] Jose Agustin Barrachina, Chengfang Ren, Gilles Vieillard, Christele Morisseau, and Jean-Philippe Ovarlez. Theory and implementation of complex-valued neural networks. *arXiv preprint arXiv:2302.08286*, 2023.

[5] Kenneth Ward Church. Word2Vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[6] Ariadne A Cruz, Kayol S Mayer, and Dalton S Arantes. RosenPy: An open source python framework for complex-valued neural networks. *SoftwareX*, 28:101925, 2024.

[7] Akira Hirose. *Complex-valued neural networks*. Springer, 2006.

[8] William W Hsieh. Nonlinear multivariate and time series analysis by neural network methods. *Reviews of Geophysics*, 42(1), 2004.

[9] Taehwan Kim and Tülay Adali. Complex backpropagation neural network using elementary transcendental activation functions. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing.*, volume 2, pages 1281–1284. IEEE, 2001.

[10] Taehwan Kim and Tülay Adali. Fully complex multi-layer perceptron network for nonlinear signal processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 32(1):29–43, 2002.

[11] Michael J Kirby and Rick Miranda. Circular nodes in neural networks. *Neural Computation*, 8(2):390–402, 1996.

[12] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[13] Yue Leng, Erik S Musiek, Kun Hu, Francesco P Cappuccio, and Kristine Yaffe. Association between circadian rhythms and neurodegenerative diseases. *The Lancet Neurology*, 18(3):307–318, 2019.

[14] Ryan W Logan and Colleen A McClung. Rhythms of life: circadian disruption and brain disorders across the lifespan. *Nature Reviews Neuroscience*, 20(1):49–65, 2019.

[15] Aram Ansary Ogholbake and Qiang Cheng. PROTECT: Protein circadian time prediction using unsupervised learning. *arXiv preprint arXiv:2501.07405*, 2025.

[16] Timothy James O'Shea, Tamoghna Roy, and T Charles Clancy. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):168–179, 2018.

[17] John G Proakis and Masoud Salehi. *Digital communications*, volume 4. McGraw-hill New York, 2001.

[18] Matthias Scholz. Analysing periodic phenomena by circular PCA. In *International conference on bioinformatics research and development*, pages 38–47. Springer, 2007.

[19] Matthias Scholz, Martin Fraunholz, and Joachim Selbig. Nonlinear principal component analysis: neural network models and applications. In *Principal manifolds for data visualization and dimension reduction*, pages 44–67. Springer, 2008.

[20] Dirk Jan Stenvers, Frank AJL Scheer, Patrick Schrauwen, Susanne E la Fleur, and Andries Kalsbeek. Circadian clocks and insulin resistance. *Nature reviews endocrinology*, 15(2):75–89, 2019.

[21] Kelsey Teeple, Prabha Rajput, Maria Gonzalez, Yu Han-Hallett, Esteban Fernández-Juricic, and Theresa Casey. High fat diet induces obesity, alters eating pattern and disrupts corticosterone circadian rhythms in female ICR mice. *PLoS One*, 18(1):e0279209, 2023.

[22] Wilhelm Wirtinger. Zur formalen theorie der funktionen von mehr komplexen veränderlichen. *Mathematische Annalen*, 97(1):357–375, 1927.

A Supplementary Material

In this section we provide Python pseudo-code for the PyTorch implementation of the circular node and for the model architectures associated with the experiments in Section 3, Section 4, and Section 5.

A.1 PyTorch-compatible circular node implementation

```
import torch
import torch.nn as nn

class CircularNodeLayer(nn.Module):
    '''
    A pytorch implementation of circular nodes from CIRCULAR
    NODES IN NEURAL NETWORKS by MICHAEL J KIRBY AND RICK MIRANDA.
    by: Audun Myers
    date: 7/24/24
    '''
    def __init__(self, input_size, output_size):
        super(CircularNodeLayer, self).__init__()
        # initialize two layers for calculating pre-state values
        self.linearN = nn.Linear(input_size, output_size)
        self.linearT = nn.Linear(input_size, output_size)

    def forward(self, I, Q):
        # get prestate values
        P_N = self.linearN(I)
        P_T = self.linearT(Q)
        # get radial values
        R = torch.sqrt(P_N**2 + P_T**2) # point wise root-square
        R = torch.clamp(R, min=1e-16)   # avoid division by zero
        # get state values for the nodes
        S_N = P_N/R
        S_T = P_T/R
        # return the state of the nodes and the magnitude
        return S_N, S_T, R
```

A.2 Model architectures: Phase-offset models for the experiments in Section 3

```
import torch
import torch.nn as nn

class CircularModel(nn.Module):
    def __init__(self, input_size, output_size):
        super(CircularModel, self).__init__()
        self.circular_layer1 = CircularNodeLayer(input_size,
            output_size)

    def forward(self, I, Q):
        outI, outQ, _ = self.circular_layer1(I, Q)
        out = torch.atan2(outQ, outI)
        return out

class LinearModel(nn.Module):
    def __init__(self, input_size, output_size):
        super(LinearModel, self).__init__()
        self.fc1 = nn.Linear(2*input_size, output_size)

    def forward(self, x, y):
        inp = torch.cat((x,y), dim=1)
        output = self.fc1(inp)
        return output
```

A.3 Model architectures: PSK demodulation models for the experiments in Section 4

```
import torch
import torch.nn as nn
import numpy as np

def bpsk_demod(input_I, sps):
    # Demodulate a BPSK (baseband) signal
    x = np.convolve(input_I,np.ones(sps),'same')[1:-1:sps]
    return (x > 0).astype(int), x

def qpsk_demod(input_I,input_Q,sps):
    # Demodulate a QPSK (baseband) signal
    I_symbols, I_sums = bpsk_demod(input_I,sps)
    Q_symbols, Q_sums = bpsk_demod(input_Q,sps)
    CodeBook = np.zeros([2,2]).astype(int)
    CodeBook[0,0] = 2
    CodeBook[0,1] = 1
    CodeBook[1,0] = 3
    CodeBook[1,1] = 0
    output_bit_string = []
    for i,j in zip(I_symbols, Q_symbols):
        output_bit_string.append(CodeBook[i,j])
    return output_bit_string, [I_sums,Q_sums]

class CircularModel_demod(nn.Module):
    def __init__(self, input_size, out1, out2, out3, output_size):
        super(CircularModel_demod, self).__init__()
        self.circular_layer1 = CircularNodeLayer(input_size, out1)
        self.circular_layer2 = CircularNodeLayer(out1, out2)
        self.circular_layer3 = CircularNodeLayer(out2, out3)
        self.linear_layer = nn.Linear(2*out3, output_size)

    def forward(self, I, Q):
        outI, outQ, _ = self.circular_layer1(I, Q)
        outI, outQ, _  = self.circular_layer2(outI, outQ)
        outI, outQ, _  = self.circular_layer3(outI, outQ)
        out = torch.cat((outI, outQ), dim=1)
        output = self.linear_layer(out)
        return output

class LinearModel_demod(nn.Module):
    def __init__(self, input_size, out1, out2, out3, output_size):
        super(LinearModel_demod, self).__init__()
        self.fc1 = nn.Linear(2*input_size, out1)
        self.fc2 = nn.Linear(out1, out2)
        self.fc3 = nn.Linear(out2, out3)
        self.linear_layer = nn.Linear(out3, output_size)
        self.relu = nn.ReLU()
        self.relu2 = nn.ReLU()
        self.relu3 = nn.ReLU()

    def forward(self, x, y):
        inp = torch.cat((x,y), dim=1)
        out = self.relu(self.fc1(inp))
        out = self.relu2(self.fc2(out))
        out = self.relu3(self.fc3(out))
        output = self.linear_layer(out)
        return output
```

A.4 Model architectures: Modulation classification models for the experiments in Section 5

```
import torch
import torch.nn as nn
```

```python

class CircularPlusCNN(nn.Module):
    def __init__(self, seq_length=128, num_classes=4,
        width_multiplier=1.0):
        super().__init__()
        self.seq_length = seq_length
        self.width_multiplier = width_multiplier
        # Calculate scalable dimensions
        circ_features = max(1, int(4 * width_multiplier))
        conv1_channels = max(1, int(20 * width_multiplier))
        conv2_channels = max(1, int(40 * width_multiplier))
        classifier_hidden = max(2, int(128 * width_multiplier))
        # Scalable circular preprocessing
        self.circular_layer = CircularNodeLayer(input_size=1,
            output_size=circ_features)
        # Scalable CNN (input channels = circ_features * 3 )
        input_channels = circ_features * 3
        self.conv_features = nn.Sequential(
            nn.Conv1d(input_channels, conv1_channels, kernel_size=8,
                stride=2, padding=4),
            nn.ReLU(),
            nn.BatchNorm1d(conv1_channels),
            nn.Dropout(0.3),
            nn.Conv1d(conv1_channels, conv2_channels,
                kernel_size=8, stride=2, padding=4),
            nn.ReLU(),
            nn.BatchNorm1d(conv2_channels),
            nn.Dropout(0.3),
            nn.AdaptiveAvgPool1d(8)
        )
        # Scalable classifier
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(conv2_channels * 8, classifier_hidden),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(classifier_hidden, num_classes)
        )
        self._initialize_circular_layer()

    def _initialize_circular_layer(self):
        with torch.no_grad():
            # Initialize circular layer
            nn.init.xavier_uniform_(
                self.circular_layer.linearN.weight)
            nn.init.xavier_uniform_(
                self.circular_layer.linearT.weight)
            nn.init.uniform_(self.circular_layer.linearN.bias,
                -0.1, 0.1)
            nn.init.uniform_(self.circular_layer.linearT.bias,
                -0.1, 0.1)
            # Initialize CNN and classifier
            for module in [self.conv_features, self.classifier]:
                for layer in module.modules():
                    if isinstance(layer, (nn.Conv1d, nn.Linear)):
                        nn.init.xavier_uniform_(layer.weight)
                        if layer.bias is not None:
                            nn.init.uniform_(layer.bias, -0.05, 0.05)

    def forward(self, x):
        batch_size, seq_len, _ = x.shape
        # Apply circular layer to each timestep
        circular_sequence = []
        for t in range(seq_len):
            i = x[:, t, 0:1]
```

```
            q = x[:, t, 1:2]
            s_n, s_t, r = self.circular_layer(i, q)
            circular_features = torch.cat([s_n, s_t, r], dim=1)
            circular_sequence.append(circular_features)
        # Create circular signal for CNN
        circular_signal = torch.stack(circular_sequence,
            dim=1).transpose(1, 2)
        # CNN processing
        conv_features = self.conv_features(circular_signal)
        return self.classifier(conv_features)

class LinearPlusCNN(nn.Module):
    def __init__(self, seq_length=128, num_classes=4,
        width_multiplier=1.0):
        super().__init__()
        self.seq_length = seq_length
        self.width_multiplier = width_multiplier
        # Match the circular layer's scaling logic
        circ_features = max(1, int(4 * width_multiplier))
        linear_output_size = circ_features * 3
        # Calculate other scalable dimensions
        conv1_channels = max(1, int(20 * width_multiplier))
        conv2_channels = max(1, int(40 * width_multiplier))
        classifier_hidden = max(2, int(128 * width_multiplier))
        # Scalable circular preprocessing
        self.circular_layer = CircularNodeLayer(input_size=1,
            output_size=circ_features)
        # Linear layer that scales with width_multiplier
        self.linear_layer = nn.Linear(2, linear_output_size)
        # CNN (input channels = linear_output_size to match circular)
        self.conv_features = nn.Sequential(
            nn.Conv1d(linear_output_size, conv1_channels,
                kernel_size=8, stride=2, padding=4),
            nn.ReLU(),
            nn.BatchNorm1d(conv1_channels),
            nn.Dropout(0.3),
            nn.Conv1d(conv1_channels, conv2_channels, kernel_size=8,
                stride=2, padding=4),
            nn.ReLU(),
            nn.BatchNorm1d(conv2_channels),
            nn.Dropout(0.3),
            nn.AdaptiveAvgPool1d(8)
        )
        # Classifier
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(conv2_channels * 8, classifier_hidden),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(classifier_hidden, num_classes)
        )
        self._initialize_weights()

    def _initialize_weights(self):
        with torch.no_grad():
            # Initialize linear layer
            nn.init.xavier_uniform_(self.linear_layer.weight)
            nn.init.uniform_(self.linear_layer.bias, -0.1, 0.1)
            # Initialize CNN and classifier
            for module in [self.conv_features, self.classifier]:
                for layer in module.modules():
                    if isinstance(layer, (nn.Conv1d, nn.Linear)):
                        nn.init.xavier_uniform_(layer.weight)
                        if layer.bias is not None:
                            nn.init.uniform_(layer.bias, -0.05, 0.05)
```

```
133
134     def forward(self, x):
135         batch_size, seq_len, _ = x.shape
136         # Apply linear transformation to each timestep
137         linear_sequence = []
138         for t in range(seq_len):
139             iq_features = x[:, t, :]
140             linear_features = self.linear_layer(iq_features)
141             linear_sequence.append(linear_features)
142         # Stack and transpose for conv1d
143         linear_signal = torch.stack(linear_sequence,
144             dim=1).transpose(1, 2)
145         # CNN processing
146         conv_features = self.conv_features(linear_signal)
147         return self.classifier(conv_features)
148
149 class BaselineCNN(nn.Module):
150     def __init__(self, seq_length=128, num_classes=4,
151         width_multiplier=1.0):
152         super().__init__()
153         self.seq_length = seq_length
154         self.width_multiplier = width_multiplier
155         # Calculate scalable dimensions
156         conv1_channels = max(1, int(20 * width_multiplier))
157         conv2_channels = max(1, int(40 * width_multiplier))
158         classifier_hidden = max(2, int(128 * width_multiplier))
159         # Scalable CNN
160         self.feature_layer = nn.Sequential(
161             nn.Conv1d(2, conv1_channels, kernel_size=8, stride=2,
162                 padding=4),
163             nn.ReLU(),
164             nn.BatchNorm1d(conv1_channels),
165             nn.Dropout(0.3)
166         )
167         self.intermediate = nn.Sequential(
168             nn.Conv1d(conv1_channels, conv2_channels, kernel_size=8,
169                 stride=2, padding=4),
170             nn.ReLU(),
171             nn.BatchNorm1d(conv2_channels),
172             nn.Dropout(0.3),
173             nn.AdaptiveAvgPool1d(8)
174         )
175         # Scalable classifier
176         self.classifier = nn.Sequential(
177             nn.Flatten(),
178             nn.Linear(conv2_channels * 8, classifier_hidden),
179             nn.ReLU(),
180             nn.Dropout(0.4),
181             nn.Linear(classifier_hidden, num_classes)
182         )
183         self._initialize_layers()
184
185     def _initialize_layers(self):
186         for module in self.modules():
187             if isinstance(module, (nn.Conv1d, nn.Linear)):
188                 nn.init.xavier_uniform_(module.weight)
189                 if module.bias is not None:
190                     nn.init.uniform_(module.bias, -0.05, 0.05)
191
192     def forward(self, x):
193         x = x.transpose(1, 2)
194         features = self.feature_layer(x)
195         intermediate = self.intermediate(features)
196         return self.classifier(intermediate)
```