uOttawa

MAT 5182 – Modern Computational Statistics

Professor : `Kelly Burkett`

Functional echo state network for time series classification

Authors:

Amine Natik – Bryan Paget – Ezekiel Williams

Submitted on : April 5th 2018

Winter term 2018

1 Abstract

For our project we aimed to reproduce the results of a recent paper that used a novel kind of neural network algorithm, a modified Echo-State Network (ESN) to classify time series data. This paper will provide an overview of our project: the introduction will give a brief background on echo-state networks, the Functional Echo State Network (FESN) and the rational for this modification of an ESN. The methods will give a mathematical description of the FESN, essentially as it is described in Ma et al. (2016). The results will compare our simulations to those of the original paper and the conclusion will summarize our findings with a few insights.

2 Introduction

The echo state network (ESN) is a three layer neural network with sparse, reccurent connections in its single hidden layer called the reservoir Jaeger and Haas (2004). The weights corresponding to the reservoir and input connections are randomly assigned and fixed. The weights of the output neurons are then optimized using linear regression so that the network can properly perform its function Montavon et al. (2012). An ESN maps vectors to vectors. A functional echo state network is an ESN that is modified to map sequences of vectors to single vectors, a necessary adaptation if one wishes to use an ESN type network to classify time series.

The recurrent echo-state connections of the FESN allow it to pick up on temporal patterns better than a feed forward network, while the simple structure makes training this network easier than training many other kinds of recurrent neural networks. We will start by talking a bit about ESN and then dive right into the FESN in the methods section.

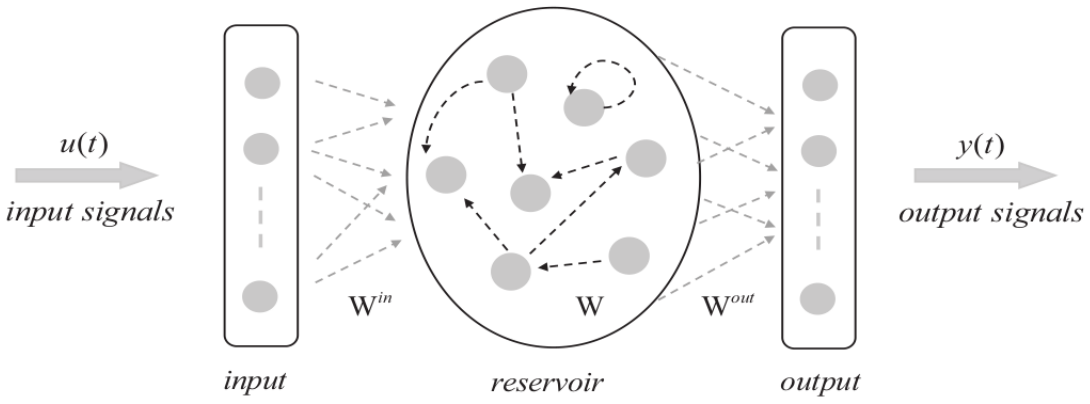

Figure 1: The architecture of an Echo State Network.

Let K denote the number of inputs, N the size of the reservoir (number of nodes in the *reservoir*), L the number of output units and T the length of time series (all time series should be of the same length). The three essential components of an echo state network are $\mathbf{W}^{\text{in}}$, an $N \times K$ matrix of input weights, $\mathbf{W}$, an $N \times N$ sparse matrix that defines the recurrent connections in the reservoir, and $\mathbf{W}^{\text{out}}$, an $L \times N$ matrix of output weights. Using these notations the ESN algorithm can be described concisely with the following two equations. For each $t = 0, \ldots, T-1$

$$\mathbf{x}(t) = f\left(\mathbf{W}\mathbf{x}(t-1) + \mathbf{W}^{\text{in}}\mathbf{u}(t)\right) \quad \text{where } \mathbf{x}(-1) := \mathbf{0} \tag{1}$$

$$\mathbf{y}(t) = f^{\text{out}}\left(\mathbf{W}^{\text{out}}\mathbf{x}(t)\right) \tag{2}$$

$\mathbf{u}(t)$ is the input signal at time t, $\mathbf{x}(t) = [x_1(t), \ldots, x_N(t)]^T$ is the state of the reservoir at time t and $\mathbf{y}(t)$ is the output at time t. The function f is used to reduce the effect of noise on the algorithm and to set upper and lower bounds on potential values of elements of $\mathbf{x}(t)$. f^{out} is typically chosen to be a linear function.

Echo state networks are usually used for time series prediction; before Ma et al. (2016), their use for time-series classification had not been fully explored. The main contribution of Ma et al. was in using functional operations to condense the output of an ESN down to a single output vector.

To achieve this, the time series at each reservoir node, $\mathbf{x}(\cdot)$, and the elements of the output weight matrix $\mathbf{W}^{\text{out}}$ are considered to be functions of time. Then summation over time integrals of the product of $x_i(t)w_{i,j}(t)$ is used to map from the space of continuous functions to the reals, yielding the desired single vector output.

3 The functional echo state network

The core idea of an FESN is to use weighted integration, a kind of "temporal aggregation", described at the end of the introduction, to map the reservoir time series from a functional field into the real number field, making the ESN a time series classifier. The use of temporal aggregation makes the task of optimizing the loss function intractable. To overcome this hurdle, we approximated the output-weight functions with orthogonal function basis expansion, which allows us to use linear regression to optimize the output weight functions in much the same way we would optimize the output weights in a standard ESN. The details of the proposed method are presented as follows.

3.1 Model Design

Figure 2 shows the structure of an FESN, with K inputs, a reservoir of size N, and L-output classes.

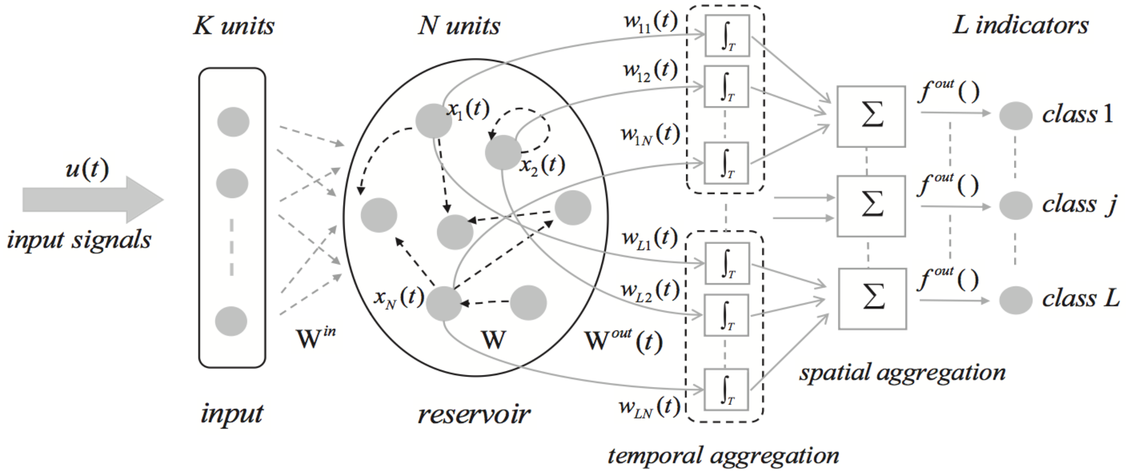

Figure 2: The general structure of FESN.

We need to randomly initialize the matrices $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$. The reservoir matrix $\mathbf{W}$ should have a given sparsity coefficient α and spectral radius ρ. After the initialization stage, the input signals $\mathbf{u}(t)$ will be input into the FESN (note that we only have a discrete sample from the signal function $\mathbf{u}(0), \ldots, \mathbf{u}(T-1)$); then, the state of the reservoir at time t, $\mathbf{x}(t) = [x_1(t), \ldots, x_N(t)]^T$ is constantly updated according to the equation (3)

$$\begin{cases} \mathbf{x}(-1) = \mathbf{0} \\ \mathbf{x}(t) = f\left(\mathbf{W}\mathbf{x}(t-1) + \mathbf{W}^{\text{in}}\mathbf{u}(t)\right), \ t = 0, 1, \ldots, T-1. \end{cases} \tag{3}$$

In the implementation we choose the activation function $f(\cdot)$ to be the Sigmoid function defined by:

$$\sigma(x) = \frac{1}{1+e^{-ax}} \quad \text{for all } x \in \mathbb{R}. \tag{4}$$

a is the single parameter value for the sigmoid function. Define $w_{ji}(t)$ for $j = 1, 2, \dots, L$ and $i = 1, 2, \dots, N$ to be the entries of the matrix $\mathbf{W}^{\text{out}}(t)$ at time t, that is:

$$\mathbf{W}^{\text{out}}(t) = \begin{bmatrix} w_{11}(t) & w_{12}(t) & \dots & w_{1N}(t) \\ w_{21}(t) & w_{22}(t) & \dots & w_{2N}(t) \\ \vdots & \vdots & \ddots & \vdots \\ w_{L1}(t) & w_{L2}(t) & \dots & w_{L1}(t) \end{bmatrix}, \tag{5}$$

using weighted integration we can rewrite Equation (2) in terms of the elements of the output vector $\mathbf{y}$ in the following way:

$$y_j = f^{\text{out}}\left(\sum_{i=1}^{N} \int_0^T w_{ji}(t)x_i(t)dt\right) \quad \text{for all } j = 1, \dots, L \tag{6}$$

We classify the time series by finding the position of the maximal value in the output L-vector $\mathbf{y} = [y_1, \dots, y_L]^T$. For this purpose we can simply set the function $f^{\text{out}}(\cdot)$ to the identity function. In the next subsection, we propose an algorithm to train the matrix $\mathbf{W}^{\text{out}}(t)$ using Fourier expansion.

3.2 Training the Model

As in the original paper, assume that N_{train} samples of the time series $\{\mathbf{u}^{(m)}(t), \mathbf{d}^{(m)}\}$ for $m = 1, 2, \dots, N_{\text{train}}$ and $t = 0, \dots, T-1$ are provided as the training set, where $\mathbf{d}^{(m)}$ is the actual class of the of the time series $\mathbf{u}^{(m)}(t)$, we will call $\mathbf{d}^{(m)}$ the teacher vector.

Under these conditions, for each $m = 1, 2, \dots, N_{\text{train}}$ the update process of the reservoir is defined by :

$$\begin{cases} \mathbf{x}^{(m)}(-1) = \mathbf{0} \\ \mathbf{x}^{(m)}(t) = f\left(\mathbf{W}\mathbf{x}^{(m)}(t-1) + \mathbf{W}^{\text{in}}\mathbf{u}^{(m)}(t)\right), \ t = 0, 1, \dots, T-1 \end{cases} \tag{7}$$

and for $j = 1, 2, \dots, L$ the jth class by:

$$y_j^{(m)} = \sum_{i=1}^{N} \int_0^T w_{ji}(t)x_i^{(m)}(t)dt \quad \text{for all } j = 1, \dots, L \tag{8}$$

therefore the FESN loss function is given by

$$F\left(\mathbf{W}^{\text{out}}(t)\right) = \min_{\mathbf{W}^{\text{out}}(t)} \sum_{m=1}^{N_{\text{train}}} \left\|\mathbf{y}^{(m)} - \mathbf{d}^{(m)}\right\|^2 \quad \text{for all } t = 0, 1, \dots, T-1 \tag{9}$$

In order to compute the minimum of the term $\left\|\mathbf{y}^{(m)} - \mathbf{d}^{(m)}\right\|^2$ the idea is to use a Fourier expansion of the functions $w_{ji}(t)$ and $x_i^{(m)}(t)$ for each $m = 1, 2, \dots, N_{\text{train}}$, $i = 1, 2, \dots, N$ and $j = 1, 2, \dots, L$. In general, if we assume that two functions $x(t)$ and $w(t)$ are T-periodic then they can be expressed in the following form

$$x(t) = \sum_{k=0}^{\infty} \alpha_k \varphi_k(t) \tag{10}$$

$$w(t) = \sum_{k=0}^{\infty} \beta_k \varphi_k(t) \tag{11}$$

where $\varphi_0, \varphi_1, \dots, \varphi_k, \dots$ is an orthonormal basis of continuous functions. As in the original paper, we use the Fourier series in Equation (12) as our basis; unlike the original paper, we only used this basis. We did not experiment with any other bases, due to time constraints.

$$\frac{1}{\sqrt{T}}, \sqrt{\frac{2}{T}} \cos\left(\frac{2\pi}{T}t\right), \sqrt{\frac{2}{T}} \sin\left(\frac{2\pi}{T}t\right), \dots, \sqrt{\frac{2}{T}} \cos\left(\frac{2\pi}{T}kt\right), \sqrt{\frac{2}{T}} \sin\left(\frac{2\pi}{T}kt\right), \dots \tag{12}$$

it is easy to check that for each $m, n \in \mathbb{N}$:

$$\int_0^T \varphi_n(t)\varphi_m(t) = \begin{cases} 1 \text{ if } m = n \\ 0 \text{ if } m \neq n \end{cases} \tag{13}$$

Now, since the signal $\mathbf{u}(t)$ is given in a discrete form (only for $t = 0, 1, \dots, T-1$) then similarly the functions $x(t)$ and $w(t)$ are defined only for $t = 0, 1, \dots, T-1$. We can not get a continuous expansion of Fourier series, hence we will only use a discrete Fourier expansion.

We define the discrete Fourier transform of the numbers $x(0), x(1), \dots, x(T-1)$ by a complex vector $\hat{X}$ of length T such that for each $k = 0, 1, \dots, T-1$:

$$\hat{X}(k) = \sum_{t=0}^{T-1} x(t) \exp\left\{\frac{-2i\pi kt}{T}\right\} \tag{14}$$

put $\alpha_k = \Re\left(\hat{X}(k)\right)$ and $\beta_k = \Im\left(\hat{X}(k)\right)$. One can check that for each $t = 0, 1, \dots, T-1$:

$$x(t) = \frac{1}{T} \sum_{k=0}^{T-1} \hat{X}(k) \exp\left\{\frac{2i\pi kt}{T}\right\} \tag{15}$$

therefore,

$$\begin{aligned} x(t) &= \frac{1}{T} \sum_{k=0}^{T-1} (\alpha_k + i\beta_k) \left(\cos\left(\frac{2\pi kt}{T}\right) + i \sin\left(\frac{2\pi kt}{T}\right)\right) && (16) \\ &= \frac{1}{T} \sum_{k=0}^{T-1} \left(\alpha_k \cos\left(\frac{2\pi kt}{T}\right) - \beta_k \sin\left(\frac{2\pi kt}{T}\right)\right) + i\frac{1}{T} \sum_{k=0}^{T-1} \left(\alpha_k \sin\left(\frac{2\pi kt}{T}\right) + \beta_k \cos\left(\frac{2\pi kt}{T}\right)\right) && (17) \\ &= \frac{1}{T} \sum_{k=0}^{T-1} \left(\alpha_k \cos\left(\frac{2\pi kt}{T}\right) - \beta_k \sin\left(\frac{2\pi kt}{T}\right)\right) \quad \textit{(since the } x(t)\textit{'s are real numbers)} && (18) \end{aligned}$$

then we can write :

$$x(t) = \sum_{k=0}^{\infty} a_k \varphi_k(t) \tag{19}$$

where the coefficients a_k are defined by:

$$\begin{cases} a_0 = \frac{a_0}{\sqrt{T}} \\ a_{2k-1} = \frac{\alpha_k}{\sqrt{2T}} & \text{for } k = 1, 2, \dots, T-1 \\ a_{2k} = -\frac{\beta_k}{\sqrt{2T}} & \text{for } k = 1, 2, \dots, T-1 \\ a_k = 0 & \text{for } k \geq 2T-1 \end{cases} \tag{20}$$

The equation (19) is the discrete Fourier expansion of the T-periodic function $x(t)$ as described in equation (10). Similarly we can get the discrete Fourier expansion of the function $w(t)$:

$$w(t) = \sum_{k=0}^{\infty} b_k \varphi_k(t) \tag{21}$$

hence, for each reservoir unit $x(t)$ and each output-weight $w(t)$ the aggregation result, using the orthonormal property of the basis $\varphi_0, \varphi_1, \ldots$ is given by :

$$\int_0^T w(t)x(t)dt = \sum_{k=0}^{\infty} a_k b_k \tag{22}$$

$$= \sum_{k=0}^{R-1} a_k b_k \tag{23}$$

where $R = 2T - 1$. Using this expansion, the equation (8) becomes :

$$y_j^{(m)} = \sum_{i=1}^{N} \int_0^T w_{ji}(t) x_i^{(m)}(t) dt \tag{24}$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{R} a_{ik}^{(m)} b_{ji(k)} \tag{25}$$

for all $j = 1, 2, \ldots, L$ and $m = 1, 2, \ldots, N_{\text{train}}$. Where $(a_{ik}^{(m)})_k$ and $(b_{ji(k)})_k$ are the coefficients defined in equation (20) for the functions $x_i^{(m)}(t)$ and $w_{ji}(t)$ respectively. Equation (9) combined with equation (25) can be rewritten as,

$$F(b_{ji(k)}) = \min_{b_{ji(k)}} \sum_{m=1}^{N_{\text{train}}} \sum_{j=1}^{L} \left(\sum_{i=1}^{N} \sum_{k=1}^{R} a_{ik}^{(m)} b_{ji(k)} - d_j^{(m)} \right)^2 \tag{26}$$

where $d_j^{(m)}$ are elements of the teacher vector $\mathbf{d}^{(m)}$. Equation (26) is solvable by the Moore-Penrose pseudo-inverse Penrose (1956). In matrix notation we have

$$\mathbf{W}^{\text{out}} = \boldsymbol{M}^{\dagger} \boldsymbol{D}. \tag{27}$$

Exactly as in Ma et al. (2016), the matrix $\mathbf{W}^{\text{out}} \in \mathbb{R}^{NR \times L}$ (above) has elements $\hat{b}_{ji(k)}$ for $j = 1, 2, \ldots, L,\ i = 1, 2, \ldots, N,\ k = 1, 2, \ldots, R$. The matrix $\boldsymbol{M} \in \mathbb{R}^{N_{\text{train}} \times NR}$ has elements $a_{ik}^{(m)}$ for $i = 1, 2, \ldots, N,\ k = 1, 2, \ldots, R,\ m = 1, 2, \ldots, N_{\text{train}}$, and $\boldsymbol{M}^{\dagger}$ is the pseudo-inverse matrix of $\boldsymbol{M}$. $\boldsymbol{D} \in \mathbb{R}^{N_{\text{train}} \times L}$, the teacher matrix, has $d_j^{(m)}$ as elements for $j = 1, 2, \ldots, L,\ m = 1, 2, \ldots, N_{\text{train}}$.

After training our FESN using basis expansion and linear regression, see `train()` on Page 7, we obtain the trained matrix $\mathbf{W}^{\text{out}}$ with entries $\hat{b}_{ji(k)}$. We are now able to classify time series, consider $\mathbf{u}^{\text{new}}(t)$ a discrete signal that we want to classify, the result of the trained FESN as in equation (25) is given by the index of the maximum element of the output vector,

$$y = \arg\max_j \mathbf{y}_j = \arg\max_j \left(\sum_{i=1}^{N} \sum_{k=1}^{R} \hat{b}_{ji(k)} a_{ik} \right) \tag{28}$$

where $(a_{ik})_k$ are the coefficients of the discrete Fourier expansion of the reservoir unit $x_i^{\text{new}}(t)$.

4 Our Implementation

We broke our code up into the following seven functions.

1. `sigmoid()`:

   Description: Reservoir activation function.

   Inputs: A real number $x \in \mathbb{R}$.

Outputs: A real number in $[0, 1]$ according to equation (4) .

```python
def sigmoid(x):
    y = 1/(1+np.exp(-0.1*x))
    return y
```

2. `fourier()`:

Description: Generate the discrete Fourier coefficients.

Inputs: A vector $\mathbf{x} = [x(0), x(1), \ldots, x(T-1)]^T$ of real numbers.

Outputs: A vector of Fourier coefficients as in equation (20).

```python
def fourier(vec):
    T=len(vec)
    coeff=np.zeros(2*T-1)
    z=np.fft.fft(vec)
    alpha=z.real
    beta=z.imag
    coeff[0]=alpha[0]/np.sqrt(T)
    for k in range(1,T):
        coeff[2*k-1]=alpha[k]/np.sqrt(2*T)
        coeff[2*k]=-beta[k]/np.sqrt(2*T)
    return coeff
```

3. `initialize()`:

Description: Generate $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$ based on specified parameters.

Inputs: Spectral radius ρ , sparsity of reservoir α, size of reservoir N, and dimension of time-series k.

Outputs: Input to reservoir connectivity matrix $\mathbf{W}^{\text{in}}$, reservoir connectivity matrix $\mathbf{W}$.

```python
def initialize(rho=0.9,alpha=0.01,N=300,k=1):

    #Define W
    W = rand(N, N, density=1-alpha).todense()# Random matrix of density 1-alpha.
    W = np.squeeze(np.asarray(W))
    eig = np.max(np.absolute(np.linalg.eig(W)[0])) # Find spectral radius of W
    scale = rho / eig
    W = scale*W  # Scale spectral radius of W so that it is equal to rho

    #Define Win
    Win = np.random.rand(N,k)

    return Win, W
```

4. `organize()`:

Description: Organize data into format that program can use.

Inputs: set of time-series to be classified (each time-series is a matrix), set of associated teacher vectors.

Outputs: matrix of teacher signals D, and the list of time-series U.

```
def organize(numclass=5,filename='Beef'):
    file=open('UCR_TS_Archive_2015/'+filename+'/'+filename+'_TRAIN','r')  # Load file
    U=list()  # Define list to hold time-series
    D=np.zeros((1,numclass))  # Predefine first row of teacher matrix

    for line in file:
        line=line.strip()  # Remove '\n' at end of line
        row=np.asarray(line.split(',')).astype(np.float)  # Set up row of u
        d=np.zeros((1, numclass))  # Predefine teacher vector length
        d[0,int(row[0]-1)]=1  # Add one to location of class in teacher vector
        row=np.delete(row,0)  # Delete first entry (class number) from row
        U.append(row)  # Add row to list U
        D=np.concatenate((D,d),axis=0)  # Add new teacher vector to the bottom of teacher matrix

    D=np.delete(D,0,axis=0)  # Delete first row (of zeros) from teacher matrix
    file.close()  # Close file

    return U, D  # Return teacher matrix and TS train list
```

5. `train()`:

   Description: The function to train the FESN.

   Inputs: The matrices $\mathbf{W}^{\text{in}}$, $\mathbf{W}$, $\boldsymbol{D}$, $\boldsymbol{U}$ (see above for definitions).

   Outputs: The trained weight functions for reservoir to output layer $\mathbf{W}^{\text{out}}$.

```
def train(Win,W,D,U):
    N_train = D.shape[0]  # Get size of training data set
    N = W.shape[0]  # Get size of reservoir
    T = U[0].shape[0]  # Get length of first time-series
    R = 2*T-1  # Number of coefficients in the Fourier expansion
    M = np.zeros((N_train,N*R))  # This matrix will contain Fourier coefficients

    # Loop through list of training time-series
    for m in range(0,N_train):
        X = np.zeros((N,T+1))  # Predefine matrix to hold time-series of reservoir activities

        for t in range(0,T):  # Loop to calculate reservoir time-series
            X[:,t+1] = sigmoid(np.dot(W,X[:,t]) + np.dot(Win,U[m,t]).reshape(N))  #Update state
        X = X[:, 1:]  # Delete first column (of zeros)

        for i in range(0,N):  #Loop to calculate Fourier transform of reservoir time-series
            M[m,(R*i:R*(i+1))] = fourier(X[i,:])  #The R coefficients of X[i,:]
    # Calculate pseudoinverse of M and multiply by D for linear regression to get Wout
    Wout = np.dot(np.linalg.pinv(M),D)
    return Wout
```

6. `TSC()`:

   Description: The function to classify time-series.

   Inputs: A new time-series $\mathbf{u}^{\text{new}}(t)$ and matrices $\mathbf{W}^{\text{in}}$, $\mathbf{W}$, $\mathbf{W}^{\text{out}}$.

   Outputs: The predicted class y of time-series $\mathbf{u}^{\text{new}}(t)$.

```
def TSC(u,W,Win,Wout):
    N = W.shape[0]  # Get size of reservoir
    T = u.shape[0]  # Get length of the time-series
    R = 2*T-1  # Number of coefficients in the Fourier expansion
    X = np.zeros((N, T+1))  # Predefine matrix to hold time-series of reservoir activities
    M = np.zeros((1,N*R))  # This matrix will contain Fourier coefficients
```

```
        for t in range(0, T):  # Loop to calculate reservoir time-series
            X[:,t+1] = sigmoid(np.dot(W, X[:,t]) + np.dot(Win, u[t]).reshape(N))  # Update state

        X = X[:, 1:]  # Delete first column (of zeros)
        for i in range(0, N):  # Loop to calculate Fourier transform of reservoir time-series
            M[1,(R*i,R*(i+1))] = fourier(X[i,:])  #The R coefficients of X[i,:]


        Y=np.dot(M,Wout)  # Calculate proxy for output layer activities
        y=np.argmax(Y)+1  # Find location in output vector of max value as the max
        #y is class, as given by model, Y is output vector
        return y, Y
```

7. `test():`

   Description: Tests the model's classification accuracy.

   Inputs: The file name for where to find test data set to use and the matrices $\mathbf{W}$, $\mathbf{W}^{\text{in}}$ and $\mathbf{W}^{\text{out}}$.

   Outputs: The error rate.

```
    def test(W,Win,Wout,filename='Beef'):
        # Organize data into a format that TSC can use
        file=open('UCR_TS_Archive_2015/'+filename+'/'+filename+'_TEST','r')  # Load file
        U=list()
        C=np.zeros((1,1))  # Predefine first row of class vector (this row will have to be deleted)

        for line in file:
            line=line.strip()  # Remove '\n' at end of line
            row=np.asarray(line.split(',')).astype(np.float)  # Set up row of u
            c=np.reshape(row[0],(1,1))  # c takes value of true class for the current time-series
            row=np.delete(row,0)  # Delete first entry (class number) from row
            U.append(row)  # Add row to list U
            C=np.concatenate((C,c),axis=0)  # Add new class number to the bottom of class vector

        C=np.delete(C,0,axis=0)  # Delete first row (of zero) from class vector
        file.close()  # Close file

        # Run TSC on every time-series in U
        y=np.zeros(len(U))  # Predefine output vector for classes given by model

        for i in range(0,len(U)):
            y[i]=TSC(U[i], W, Win, Wout)[0]

        # Compare TSC classifications (y) to true classes (C) element-wise and mean for error
        ER=1-np.mean(np.equal(y,C.flatten(order='C'))*1)

        return ER, y, C
```

4.1 Execution

The following code was used to call the above functions to initialize the matrices $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$, organize the data for input, train $\mathbf{W}^{\text{out}}$, and compute the error `ER`.

```
f = sigmoid  # Define f as the sigmoid function
Win,W = initialize(rho=0.9,alpha=alpha,N=n,k=1)  # Initialize first two matrices Win=W0[0], W=W0[1]
U,D = organize(numclass=15,filename='SwedishLeaf')  # Organize data: U=data[0], D=data[1]
Wout = train(Win,W,D,U,f,a=0.01)  # Train model
ER,y,C = test(W,Win,Wout,f,filename='SwedishLeaf', a=0.01)
```

5 Experimental Results

5.1 Performance on Specific Data Sets

We primarily tested our implementation on the following UCR data sets, Beef, Swedish Leaf, and Adiac Chen et al. (2015). We chose the aforementioned data sets because they had a small (5), medium (15), and large (37) number of classes, respectively. We performed small grid search experiments over hyperparameter values N, a, and α. The results from the beef data set show that our model qualitatively reproduces the behavior of Ma et al..

5.1.1 Beef

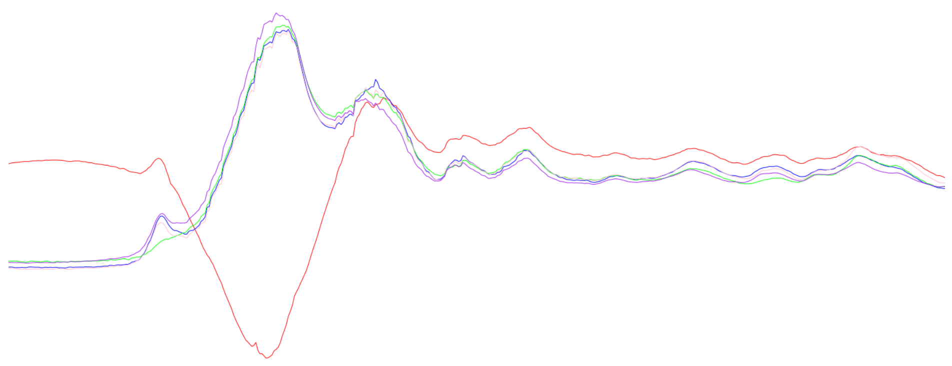

Figure 3: Beef

The beef dataset consists of five classes of beef spectrograms, from pure beef and beef contaminated with different amounts of organ meat. See Figure 3 for a graphical depiction of the data. Ma et al. tested their FESN on the beef data set using many different hyperparameter settings. They found that N did not make much difference but classification performance was affected by the values of α (the sparcity of the reservoir) and a (the parameter for the activation function). See Figure 4 for details.

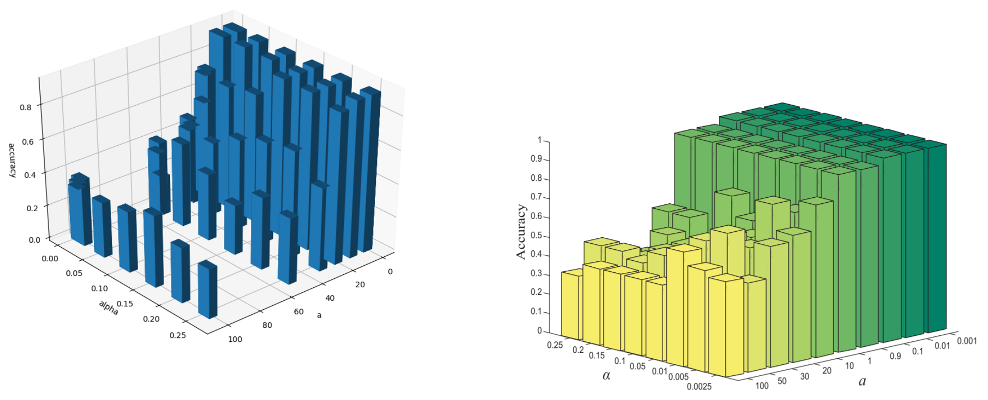

Figure 4: Our Results Reproduce Ma et al.

It was observed by Ma et al., that when a is in the interval [0.001, 1], the FESN achieves nearly the same accuracy regardless of the value of N (size of reservoir), α (sparcity of reservoir), and ρ (spectral radius of reservoir). We interpret this to mean when the activation function is less sensitive, the echo state network is less bothered by noise, and therefore the FESN is using only the most fundamental parts of the time-series for classification, at least on this data set. Classification performance was good, with a maximal classification accuracy of 93.4%. Ma et al. achieved perfect accuracy on this data set.

5.1.2 Swedish Leaf

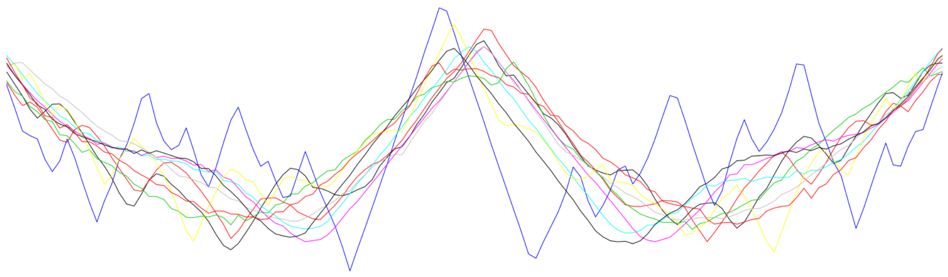

Figure 5: Swedish Leaf

Swedish leaf is a set of leaf outlines with 15 different classes. This data set was created by Oskar Söderkvist for his MSc. thesis in computer vision, see Figure 5. Figure 6 shows how the time series data are created.

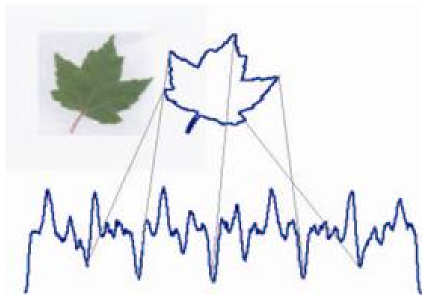

Figure 6: Swedish Leaf (image courtesy of Chen et al. (2015))

We tested our implementation of the FESN on the SwedishLeaf data set, the results were less interesting. Classification performance was unaffected by the settings for α and N (see Figure 7). The best result was 66.24% accuracy. Ma et al. achieved accuracy of 89.7% on the same data set.

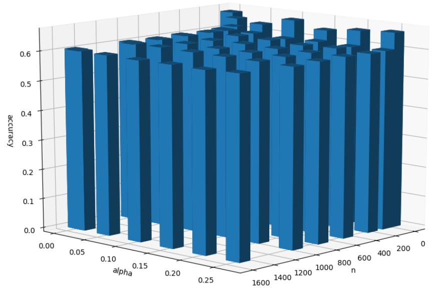

Figure 7: Swedish Leaf Results

5.1.3 ADIAC

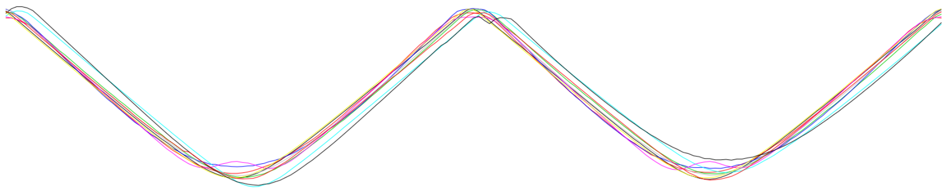

Figure 8: ADIAC

The Automatic Diatom Identification and Classification (ADIAC) project aimed to automatically identify of diatoms (unicellular algae) from microscopic outlines extracted from thresholded images. The images were transformed into time series by recording the distance to a reference point.

Figure 9: Unicellular Algae (image courtesy of Chen et al. (2015))

We chose this data set because it has 37 different classes and Ma et al. were unable to achieve a good classification accuracy (61.6% accuracy), so we wanted to see how our implementation would fare on this difficult task. Our implementation was only able to achieve 41.94% accuracy.

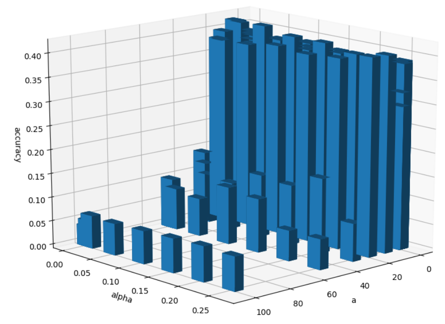

Figure 10: ADIAC Experimental Results

6 Conclusion

Our experimentation shows that our FESN implementation succeeds at qualitatively reproducing the results of the original FESN program. Figure 4 indicates that our FESN responds to parameter changes in the same way as the original. Comparison of results on the three tested data sets show that our FESN exhibited poorer classification accuracy on data sets with more classes, just like the original. However, our classification accuracy was lower than that of the original program. We have two hypotheses for why this might be.

First, our function used a Fourier expansion of length equal to $2T-1$, where T is the length of the time series. This was done for ease of Fourier computation using Python's `numpy.fft` (Fast Fourier Transform) algorithm. In the original program they experimented with changing the length of the Fourier expansion and using other kinds of orthogonal expansions (e.g. wavelet transform). It could be that by using a different length or a different function expansion Ma et al. (2016) were able to increase accuracy. Second, the authors did not explain precisely how they initialized the fixed matrices, $\mathbf{W}^{\text{in}}$ and $\mathbf{W}$, in their algorithm. Since choice of matrix initialization plays a significant role in the accuracy of a neural network they could have obtained greater accuracy simply by using a matrix initialization different than ours.

In closing, based on our results and those of Ma et al. (2016), it appears that FESN could potentially have been a competitive time series classifier at the time of its inception. However its high variability in classification accuracy across data sets and its poor ability to classify many of the test sets that had large numbers of classes make FESN appear to be a poor option, in our eyes, for time series classification in most real-world scenarios.

References

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., and Batista, G. (2015). The ucr time series classification archive. www.cs.ucr.edu/~eamonn/time_series_data/.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80.

Ma, Q., Shen, L., Chen, W., Wang, J., Wei, J., and Yu, Z. (2016). Functional echo state network for time series classification. *Information Sciences*, 373:1 – 20.

Montavon, G., Orr, G. B., and Muller, K.-R. (2012). *Neural Networks: Tricks of the Trade*. Lecture notes in computer science ; 7700. Springer Berlin Heidelberg, Berlin ; New York.

Penrose, R. (1956). On best approximate solutions of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52(1):17–19.