MODELLING TECHNIQUES FOR ELECTRICITY LOAD FORECASTING

DISSERTATION/THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY
IN
POWER SYSTEMS

Submitted by:

VEDANSHU

2K17/PSY/20

Under the supervision of

M.M. Tripathi

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

2019

DEPARTMENT OF ELECTRICAL ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I, **VEDANSHU**, Roll No. **2K17/PSY/20** student of M. Tech. (Power System), hereby declare that the dissertation/project titled "**Modelling Techniques for electricity load forecasting**" under the supervision of **M.M. Tripathi** of Electrical Engineering Department Delhi Technological University in partial fulfillment of the requirement for the award of the degree of Master of Technology has not been submitted elsewhere for the award of any Degree.

Place: Delhi **VEDANSHU**

Date: 30.07.2019

M.M. Tripathi

Professor Electrical Engineering

ABSTRACT

Single layer Feedforward Neural Network(FNN) is used many a time as a last layer in models such as seq2seq or a simple RNN network. The importance of such layer is to transform the output to our required dimensions. When it comes to weights and biases initialization, there is no such specific technique that could speed up the learning process. We could depend on deep network initialization techniques such as Xavier or He initialization. But such initialization fails to show much improvement in learning speed or accuracy. Zero Initialization (ZI) for weights of a single layer network is proposed here. We first test this technique with on a simple RNN network and compare the results against Xavier, He and Identity initialization. As a final test we implement it on a seq2seq network. It was found that ZI considerably reduces the number of epochs used and improve the accuracy. Multi-objective swarm intelligence is also utilized for weights and biases initialization for quicker learning. The developed model has been applied for short-term load forecasting using the load data of Australian Energy Market. The model is able to forecast the day ahead price accurately with error of 0.94%.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Forecsating is a very important part of business, whether it's forecasting of load or revenue of a company. With proper techniques, events can be forecasted with an accuracy under 1%. Improving the accuracy of the model helps in saving millions of dollar [1]. For example let's suppose for long term load forecasting a utility with 1 gigawatt annual peak load, the risk of oversizing and undersizing would be $0.01 * 1,000MW = 10MW$. Assuming the capital cost of $\$10,000/KW$, the overnight capital cost would be $\$10,000/KW * 10MW = \$10$ million. The saving of deferring \$10 million in spending for 1 year with 5 % interest rate would be $\$10\text{million} - \$10\text{million}/(1+0.5) = \$476,000 \approx \$500,000$. If the utility uses forcasting for obtaining energy form day ahead market, they might save around $\$300,000$ per year by improving the accuracy of 1 %.

Many methods for time-series forecasting have been proposed till now. One of the earliest and accurate method utilized a combination of time-series and regressional approaches[2]. But these type of model lakes memory, so they were not able to store useful information. One of the earliest research in this area was the use of recurrent backpropogation [3] but was time consuming. Later Long Short Term Memory cells (LSTM) was introduced [3] which solved complex, artificial long time lags tasks. A variant of LSTM which is widely used in time-series forecasting is Gradient Recurrent Unit (GRU) [4] greatly reduced the number of variables involved.

The problem of vanishing and exploding gradiants was still present in deeper networks which was fixed by the utilization of *relu* and Xavier initialization [5]. Another variant of Xavier is He [6] initialization. Identity Recurrent Neural Network (IRNN) [7] is also a variant of RNN widely used in which an identity matrix is used for the initialization purpose.

One of the problem with RNN is that the input length has to be equal to the output length, so if there are n input features then there would be n output features. But that might not be the requirement in all the use cases. Sometimes the number of output might have to be in condensed form. For that generally the output of RNN is connected to a Dense network (a layer with its all element connected to the every input features) but with identity activation and zero biasing (weighted summation). But still the weight of that layer need to be learned. This added layer could be also be used as an extra hidden layer with proper type of activation and initialization. This paper proposes one such type of initialization technique that could be used at this dense layer for faster convergence speed in training. To prove this hypothesis, a simple Feed-forward Neural Network with 1 hidden layer having 10 units was trained. The results shown by that

1.1 MODEL

By training the model, we try to approximate the output to the function $g(z)$ for every input z. The learning of weights and biases conventionally deals with the minimizing of an objective/loss function through regression. It's quantification can be done by defining a cost function as:

$$L(w,b) \equiv \frac{1}{2N} \sum_{z} \parallel g(z) - o \parallel^2 \tag{1.1}$$

where, following notations are used:

1. w : weights
2. b : biases
3. N : training input
4. o : output vector
5. z : input vector

$\parallel u \parallel$ denotes the length of the vector u. L is the cost/loss function, here, quadratic cost function, also referred as mean squared error (MSE). Inspection of the cost function shows that $L(w,b)$ is non-negative. Also,

as the output o for all the training input z is approximated by $g(z)$, the cost $L(w,b)$ starts decreasing to zero, i.e., $L(w,b) \approx 0$. For minimizing the cost function, a set of weights and biases were found using two algorithms, namely gradient descent and chicken swarm algorithm. First the training was done using only gradient descent next the training was done using a combination of chicken swam and gradient descent.

The use case of the quadratic cost function might seem random. Choosing a different cost function would have given different result of weights and biases. The next section of this paper, a set of objective function are introduced which would be then minimized collectively.

1.1.1 Gradient Descent Algorithm

If L is the function of n variables, $u_1, u_2, \ldots, u_n$, then a small change in $\Delta u = (\Delta u_1, \ldots, \Delta u_n)^T$ will lead to a change ΔL in L.

$$\Delta L \approx \nabla L \cdot \Delta u \tag{1.2}$$

where,

$$\nabla L \equiv \left(\frac{\partial L}{\partial u_1}, \cdots, \frac{\partial L}{\partial u_n} \right)^T \tag{1.3}$$

In equation 1.2, ∇L relates the changes in u to the changes in L. If Δu is chosen in the following way:

$$\Delta u = -\eta \nabla L \tag{1.4}$$

where, η denotes a small positive learning rate. Then from equation 1.2 and 1.4 it can be written that $\Delta L \approx \eta \nabla L \cdot \nabla L = -\eta \parallel \nabla L \parallel^2$. Since, $\parallel \nabla L \parallel^2 \geq 0$ it is guaranteed that $\Delta L \leq 0$. Following update rule is applied repeatedly to minimize the cost function.

$$u \rightarrow u' = u - \eta \nabla L \tag{1.5}$$

Above procedure defines the gradient descent algorithm (GDA). The GDA is used in neural network to minimize the cost function by finding a appropriate set of weights and biases 1.1. The update rule given in equation 1.5 is written for the weights and biases as follows:

$$
\begin{aligned}
w_k \to w'_k &= w_k - \eta \frac{\partial L}{\partial w_k} \\
b_l \to b'_l &= b_l - \eta \frac{\partial L}{\partial b_l}
\end{aligned}
\tag{1.6}
$$

Now, m number of small training inputs are randomly selected, $Z_1, Z_2, \ldots, Z_m$, referred as a mini-batch.

The average value of ∇L_{Z_j} is approximated to the overall average of ∇L_z with the assumption that the size of mini-batch is large enough, i.e,

$$
\frac{\sum_{j=1}^{m} \nabla L_{Z_j}}{m} \approx \frac{\sum_z \nabla L_z}{n} = \nabla L \tag{1.7}
$$

or re-writing the 1.7 as,

$$
\nabla L \approx \frac{1}{m} \sum_{j=1}^{m} \nabla L_{Z_j} \tag{1.8}
$$

The equation 1.8 confirms that the overall gradient can be computed by just calculating the gradient of a mini-batch. This defines the stochastic gradient descent algorithm.

Applying stochastic gradient descent algorithm to neural network, the equation 1.6 can be updated as follows:

$$
\begin{aligned}
w_k \to w'_k &= w_k - \frac{\eta}{m} \sum_j \frac{\partial L_{Z_j}}{\partial w_k} \\
b_l \to b'_l &= b_l - \frac{\eta}{m} \sum_j \frac{\partial L_{Z_j}}{\partial b_l}
\end{aligned}
\tag{1.9}
$$

the summation in the equation 1.9 is within the current mini-batch over all the training examples Z_j. One epoch is said to be completed when all the training mini-batches have been picked.

1.1.2 Multi-objective optimization

An objective function(vector valued) is defined as follows:

$$
g \colon Z \to R^k, g(z) = (g_1(z), \ldots, g_k(z))^T \tag{1.10}
$$

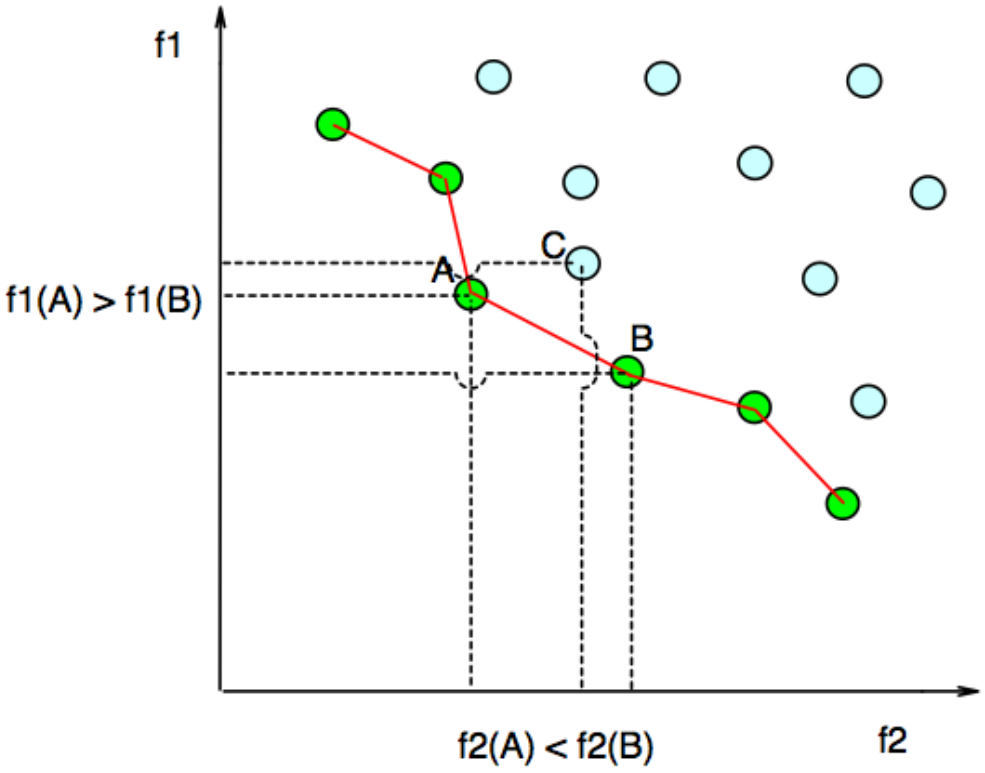


Figure 1.1: The red line shows the Pareto frontier. Circles denote the feasible choices. Smaller values are preferred for the minimizing the objective function.

A multi-objective optimization problem is formulated as,

$$min(g_1(z), g_2(z), \ldots, g_k(z)) \tag{1.11}$$

For an element $z^* \in Z$ called feasible solution, the output vector $v^* := g(z^*)$ is called an objective vector. Typically, a feasible solution which minimizes all the objective functions simultaneously does not exist. Hence, Pareto optimal solutions are found such that the solution of any of the objective function can't be improved without degrading other objective function solutions (shown in figure 1.1). In the figure 1.1 point C is not on the Pareto frontier because it is dominated by by both A and B.

A feasible solution $z^1 \in Z$ dominates another solution $z^2 \in Z$, if

1. $g_i(z^1) \leq g_i(z^2)$ for all indices $i \in \{1, 2, \ldots, k\}$ and
2. $g_j(z^1) < g_j(z^2)$ for at least one index $j \in \{1, 2, \ldots, k\}$.

For forecasting neural network the multi-objective functions are defined as follows:

$$O_1 = \frac{1}{N}\sum_{i=1}^{N}|\hat{z}_i - z_i| \tag{1.12}$$

$$O_2 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{z}_i - z_i)^2} \tag{1.13}$$

$$O_3 = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{z_i - \hat{z}_i}{z_i}\right| \tag{1.14}$$

$$O_4 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\left|\frac{z_i - \hat{z}_i}{z_i}\right| - \frac{1}{N}\sum_{i=1}^{N}\left|\frac{z_i - \hat{z}_i}{z_i}\right|\right)^2} \tag{1.15}$$

where $\hat{z}_i$ is the forecasted value and z_i is the observed value.

First a gnome was created with the combined size of weights matrix and biases matrix of the ANN. For example if the ANN was chosen with input size of 36, 10 hidden layers and output of 46. This will make two weights matrix of combined size of $10*36+46*10$ and biases matrix of combined size of $10+46$.So, the size of the gnome would be $10*36+46*10+10+46$.

CHAPTER 2: SEARCH FOR THE BEST SWARM INITIALIZER

Short-term electricity forecasting is particularly critical for electric utilities [23]. Load forecasting is essentially incorporate in planning and operations of power system. It is also used for income projection, rate plan, energy trading, et cetera[10].

ANN models are an elective method for load forecasting, and, as far as precision, contend straightforwardly with the models referred to above. Forecasting using prediction intervals based neural network[16], entropy-based feature selection in combination with soft computing techniques [17] and others[18][19] are a portion of the ongoing work done in the field of forecasting utilizing ANN.

The principle target was to test the new initialization technique and compare it with other techniques commonly used rather than improving the accuracy of forecasting.

2.1 OVERFITTED NEURAL NETWORKS

Physicist Enrico Fermi once commented " I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk." [21] . It make sense since a model with an expansive number of free variables can portray an extensive variety of phenomenons. Even though that model might show good results for the given dataset it might fail to work for new datasets. The problem with more free variables is that in most of the cases it fails to generalize the phenomenon.

ANN has made a reputation of being overfitted [12] because of use of a large number of parameters but its hard to comment whether an ANN is over-parameterised or not[20]. One of the technique used to avoid overfitting is *early stopping* but requires some judgmental to de-

cide when to stop. It requires to stop the training when we have reached saturation in accuracy.

Another technique used to avoid overfitting is *L1 regularization*. This penalizes the cost function when the weight is large. The cost function would look like this

$$C = C_0 + \frac{\lambda}{n} \sum_w |w| \tag{2.1}$$

The update rule then looks like

$$w' = w - \frac{\eta \lambda}{n} \text{sgn}(w) - \eta \frac{\partial C_0}{\partial w} \tag{2.2}$$

where, sgn(w) is the sign of the weight and λ is the *regularization parameter*. There is also L2 regularization technique which penalizes cost proportional to weight. While L1 regularization penalizes the weights by a consistent sum towards zero. In this paper we have used L1 regularization because the ANN is first initialized using our optimization algorithm, so it make sense that we might not be getting large weights in our training. But still it was a matter of choice.

2.2 DATASET

Aggregated demand data was collected between 2015 and 2017. The Australian states chosen for the datasets were Queensland (QLD), South Australia (SA), New South Wales (NSW), Victoria (VIC) and Tasmania (TAS). The electricity demand data were recorded for every half hour. It was assumed that the forecast is issued after every 12 hours for a forecasting horizon of 48 hours and a half hourly temporal resolution.

The preprocessing of data started with the normalization of the dataset. Next step was to create validation sets. The validation sets were created using two approaches. The first approach was to split the training and validation data into sets of 84 consecutive hours ($36+48$) and those sets were used to define the 5-fold validation splitting. Fold 1 contained sets 1, 6, 11 and so on as validation set and the remaining as training set. Fold 2, contained 2, 7, 12, ..., as validation, and analogous splitting were done for folds 3, 4 and 5.

SET n	36 hours data (Starts 00:00) 48 hours data (Starts 12:00)
SET n+1	36 hours data (Starts 12:00) 48 hours data (Starts 00:00)

Another approach used was a variation on KFold. In the k^{th} split, it returns first k folds as train set and the $(k+1)^{th}$ fold as test set. An example of splitting into 5 folds would look like as follows:

```
TRAIN: [0] TEST: [1]
TRAIN: [0 1] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
TRAIN: [0 1 2 3] TEST: [4]
TRAIN: [0 1 2 3 4] TEST: [5]
```

2.3 GAUSSIAN DISTRIBUTION FOR INITIALIZATION

For making an ANN graph we have to introduce the weights and biases. For the greater part of the cases we utilize random Gaussian variable with unit standard deviation and zero mean. Let's suppose that we have got an ANN graph with 500 input neurons. The weight matrix connecting the neurons of input layer to the hidden layer is instated utilizing normalized Gaussian distribution. Now a input is applied with half of them set to one and half of them set to zero. Then the weighted sum of the first neuron of the hidden layer would be $z = \sum_j w_j x_j + b$. 250 of the terms in the weighted sum would vanish as half of the input is set to zero. So, z is the weighted sum of 251 normalized Gaussian random variables including 1 extra bias term. Thus z is disseminated with zero mean and $\sqrt{251} \approx 15.84$ standard deviation . Thus z has a very broad Gaussian distribution. This implies that $|z|$ would be expansive, i.e., $z \gg 1$ or $z \ll -1$. Let's suppose we have used sigmoid (σ) as the activation function of the ANN graph, since z is large the, $\sigma(z)$ would be either zero or one. Meaning the hidden neuron is saturated. In this manner rolling out little improvements in the weight matrix would roll

out little improvements in the activation of the neuron and further those little changes in the activation would scarcely influence other neurons of the network and we will see little changes in the cost function. Thus our graph would learn gradually with the use of gradient descent algorithm. This can be understood well with the four equations of backpropagation:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{2.3}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2.4}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{2.5}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{2.6}$$

Where the error δ^l_j of neuron j in layer l is defined as the partial derivative of cost w.r.t. input weighted sum (z^l_j) of neuron j of layer l:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j} \tag{2.7}$$

$\nabla_a C$ is defined to be a vector whose components are the partial derivatives of cost w.r.t. activation of last layer $\partial C / \partial a^L_j$ and $\odot$ is the Hadamard product[3]. Similarly, $(w^{l+1})^T$ is the transpose of the weight matrix for the $(l+1)$ layer.

One way to overcome this problem would be to make the Guassian distribution sharply peaked. Assume, for a neuron we have n_{in} weights then we instate those weights as Gaussian random variables with zero mean and $1/\sqrt{n_{in}}$ standard deviation. Bias would still be initialized as a Guassian with zero mean and unit standard deviation because it's much less likely that our neuron would saturate. We could likewise initialize the biases to zero and let the gradient descent take in the appropriate biases. Suppose we have 500 inputs to the graph with half of them as zero and other half as one. Then the standard deviation of the weighted

sum of the neuron j of the hidden layer would we calculated as

$$\begin{aligned} var(z_j) &= var\left(\sum_{i=1}^{500} w_i x_i + b\right) \\ &= var\left(\sum_{i=1}^{500} w_i x_i\right) + var(b) \\ &= \sum_{i=1}^{500} var(w_i x_i) + var(b) \\ &= \sum_{i=1}^{250} var(w_i) + var(b) \\ &= \frac{250}{500} + 1 \\ &= \frac{3}{2} \end{aligned}$$

This demonstrates that z has the Gaussian distribution with zero mean and standard deviation of $\sqrt{3/2} = 1.22$ which is more pointedly crested than previously.

2.4 CUCKOO SEARCH FOR INITIALIZATION

We started with minimizing the following multiobjective functions:

$$ob_1 = \frac{1}{N}\sum_{i=1}^{N} |\hat{x}_i - x_i| \tag{2.8}$$

$$ob_2 = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\hat{x}_i - x_i)^2} \tag{2.9}$$

$$ob_3 = \frac{1}{N}\sum_{i=1}^{N} \left|\frac{x_i - \hat{x}_i}{x_i}\right| \tag{2.10}$$

$$ob_4 = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left(\left|\frac{x_i - \hat{x}_i}{x_i}\right| - \frac{1}{N}\sum_{i=1}^{N} \left|\frac{x_i - \hat{x}_i}{x_i}\right|\right)^2} \tag{2.11}$$

where $\hat{x}_i$ is the forecasted value and x_i is the observed value. This was a multiobjective problem with finding $min(ob_1, ob_2, ob_3, ob_4)$. Cuckoo search algorithm was found to be better in finding the minimum of such

type of problems[4].

First a gnome was created with the size equal to the sum of size of weights matrix and biases matrix of the ANN. For example if the ANN was chosen with input size of 36, 10 hidden layers and output of 46. This will make two weights matrix of combined size of $10 * 36 + 46 * 10$ and biases matrix of combined size of $10 + 46$.So, the size of the gnome would be $10 * 36 + 46 * 10 + 10 + 46$. The size of the gnome along with the multiobjective function $min(ob_1, ob_2, ob_3, ob_4)$ was given to the cuckoo search algorithm.

2.4.1 Mathematical model

The aim of the CSO is to find the best solution in the form of eggs in a nest. A new best solution replaces the less good solution [22]. Typically a nest contains only a single egg. The algorithm is as follows:

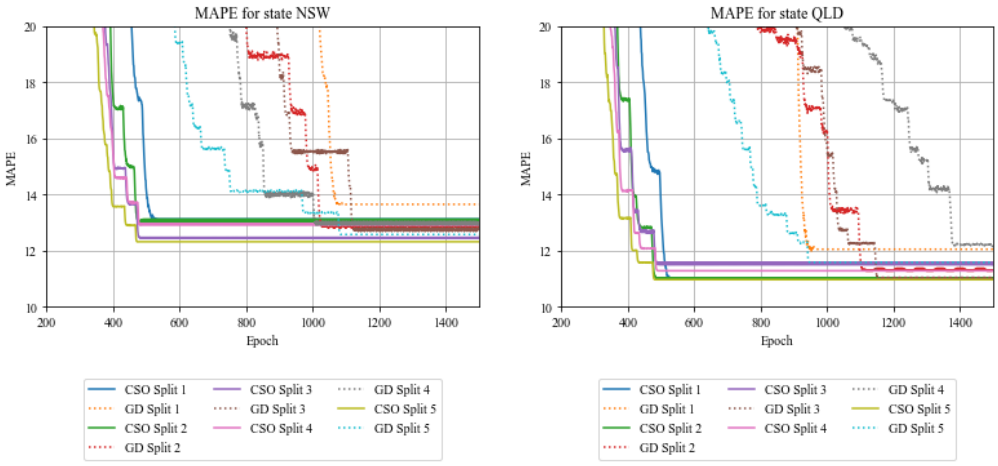
1. Eggs are laid randomly in a nest
2. High quality eggs are passed to next generation
3. The probabilty of finding a egg by the nest owner is $\Xi_a \in (0,1)$ and removed from the nest. The CSO algorithm scheme could be described in the following form:
4. Initialize the population $S = \{s_i, i \in [1 : |S|]\}$ from $|S|$ foreign nests and a cuckoo, i.e. define the initial values of for vector components $X_i = \{x_i i \in [: |S|]\}$ and cuckoo's initial position vector X_C;
5. Make a number of cuckoo's random moves in the search space by performing Levy flights and find the new cuckoo's position X_C;
6. Randomly pick a newt $s_i, i \in [1 : |S|]$ and if $f(X_C) > f(X_i)$ then substitute an egg in this nest to the cuckoo's egg, i.e. $X_i = X_C$;
7. With the probability Ξ_a remove a number of the worst randomly chosen nests (including probably s_i nest) from population and create the same number of new nests according to the 1st step rules;
8. Until the stop condition is not satisfied, proceed to the 2nd step.

2.4.2 Algorithm

```
BEGIN
  Create beginning populace of n
  nests xj, (j = 1, 2, ... ,n)

  REPEAT
    Place cuckoo to point xi randomly
    by performing Levy flights

    Nest j is chosen among n nests randomly

    IF Fj > Fi
      xj is replaced by new solution
    END IF

    Erase from the populace nests
    found with pa probability and
    construct a similar number of new
    nests

    SAVE best solution (nest)
  UNTIL stop criteria
  Postprocess results and visualization
END
```

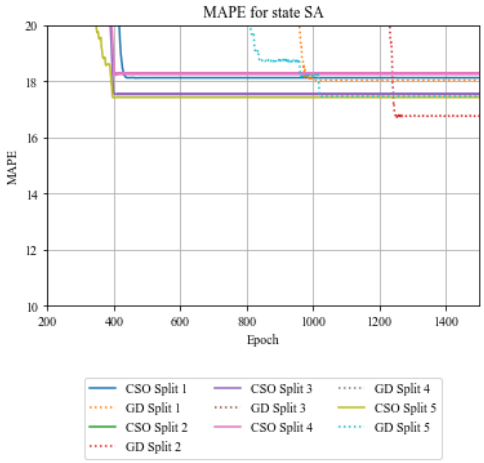
The probability of cuckoo's egg detection was chosen to be 0.25 and number of nests were 100. The gnome passed by the above algorithm was tested with every iteration against a pair of input and output vector selected randomly from the dataset. After 500 iterations the gnome was passed to the ANN for the initialization of weights and biases.

From the fig 2.1 it is clear that the CSO is dominating the results. If we see the graph of state NSW in fig 2.1(a), it can be seen that with CSO, the training is getting saturated at around 500 epochs. While with the Gaussian distribution, the training is getting saturated at around 1100 epochs. With this huge difference in the number of epochs motivated us

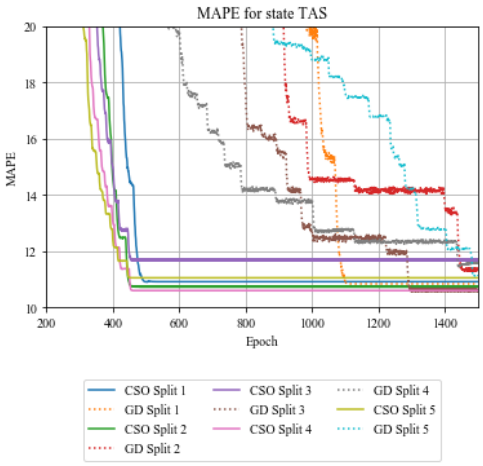


(a)

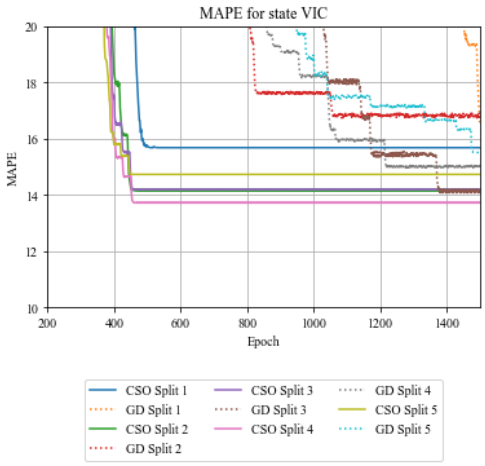
(b)



(c)



(d)



(e)

Figure 2.1: Graphs showing training convergence with both types of initialization for states: (a) NSW; (b) QLD; (c) SA; (d) TAS; and, (e) VIC.

to test other swarm intelligence for the initialization purpose.

2.5 BACTERIAL FORAGING OPTIMIZATION

2.5.1 Description

By observing the behaviour of Escherichia coli (next E.coli), Passino proposed the Bacterial Foraging Optimization [24]. A set of tensile flagella helps in locomotion during foraging. Tumbling and swimming is achieved with the help of Flagella during foraging. During the rotation of flagella (clockwise), each flagellum pulls on the cell which helps in the motion of flagella. By moving the flagella in counter-clockwise, the bacterium can swim much faster. During chemotaxis, the bacteria likes to move towards a nutrient gradient and avoid noxious environment. Their length is increased after getting sufficient food. The bacteria also, undergo reproduction by splitting itself in the middle. The chemotactic progress may be destroyed due to sudden occurrence of environmental change or attack which may lead to the motion of a group of bacteria to some other places or other groups may be introduces in the swarm. This may lead to a event of elimination-dispersal where all the bacteria in a region are eliminated or a group is dispersed into a new part of the environment.

Bacterial Foraging Optimization has three main steps:

1. Chemotaxis
2. Reproduction
3. Elimination and Dispersal

2.5.2 Mathematical model

Chemotaxis

The movement of an E.Coli bacteria is reproduced through swimming and tumbling by means of flagella. There are two method for movement of an E.Coli, one includes swimming for a timeframe toward a path and different includes tumbling. Microscopic organisms shifts back and forth between these two modes for the whole lifetime.

Reproduction

Unhealthy microscopic organisms in the end die while the healthy microorganisms (having lower objective function value lives) recreate by parting itself into two half. The reproduced bacteria stays in a similar area which aides in keeping the swarm size steady.

Elimination and Dispersal

One of the environmental changes is temperature rise which may lead to elimination of a group of bacteria in a region having a high nutrient concentration. In a different case a group might disperse into a new location. For the reproduction of this procedure a few microbes are liquidated with a little likelihood. The new substitution are arbitrarily introduced over the search space.

2.5.3 Algorithm

```
BEGIN
 Initialize randomly the bacteria foraging
   optimization population
 Fitness value of each agent is calculated
 Set global best agent to best agent
 FOR number of iterations
  FOR number of chemotactic steps
   FOR each search agent
    Move agent to the random direction
    Calculate the fitness of the moved agent
    FOR swimming length
     IF current fitness is better than previous
      Move agent to the same direction
     ELSE
      Move agent to the random direction
     END
    END
   END
   Fitness value is calculated
```

```
  END
  Fitness value of all chemotactic loops (
    health of agent) are computed and sorted
  Half of the population is splitted according
    to the health
  IF NOT last iteration
   FOR each search agent
    With some probability replace agent with
      new random generated
   END
  END
  Best search agent is updated
 Fitness value of each agent is calculated
END
```

2.6 GRAY WOLF OPTIMIZATION

2.6.1 Description

The Gray Wolf Optimization algorithm mimics the leadership hierarchy and hunting mechanism of gray wolves in nature [25]. Wolves live in a pack. The average pack consists of a family of 5–12 animals. wolves have strict social hierarchy which is represented by the division of a pack into four levels: alpha, beta, delta, and omega.

Alpha wolves are the leaders of their pack. They are responsible for making decisions, but sometimes alphas can obey to other wolves of the pack.

Beta wolves help alphas make decisions, every beta is a candidate to become an alpha if an alpha has died or aged. A beta respects an alpha and transfers commands to the pack, ensures discipline among inferior wolves and provides a feedback from the pack to an alpha.

Delta wolves are submitted by alphas and betas, but dominates the omega.

Finally, *omega* wolves have to obey all other wolves. Sometimes they play a role of caretakers.

Gray wolf hunts its prey in three main phases:

1. Tracking, chasing, and approaching
2. Pursuing, encircling, and harassing
3. Attacking

2.6.2 Mathematical model

The social hierarchy of the wolves is chosen by the fitness value. The wolf having the fittest value is the alpha, in this way the second and third fittest values are the beta and delta wolves. Rest are the omegas. Hunting (here optimization) is lead by the alpha, beta and delta. Omega follows alpha, beta and delta.

2.6.3 Algorithm

```
BEGIN
  Initialize randomly the gray wolf population
  Find 1st, 2nd and 3rd best agents alpha, beta
    , delta
  Set global best agent to the 1st best agent
  Fitness value is calculated for each search
    agents
  WHILE count < number of iterations
    FOR each search agent
    Position of each search agent is updated
    END
    Update alpha, beta and delta
    Fitness value of each search agent is
      calculated
    Update the best search agent, the 2nd best
      search agent, and the 3rd best search
      agent
    ADD 1 to count
  END WHILE
  RETURN the best search agent
END
```

2.7 BAT ALGORITHM

2.7.1 Description

The Bat Algorithm is based on the bats echolocation ability [26]. By using echolocation bats can detect their food and preys and even distinguish between the different kinds of insects in the darkness. A bat emits a loud sound and listens to the echo which is created from the sound reflection from the surrounding objects. Sounds emitted by a bat are vary in properties and can be used depending on the hunting strategy. Each sound impulse lasts from 8 to 10 milliseconds and has constant frequency between 25 and 150 KHz. A bat can emit from 10 to 20 of supersonic impulses per second, an impulse lasts from 5 to 20 milliseconds. The number of signals emitted by a bat can be increased during a hunt to 200.

2.7.2 Mathematical model

The Bat Algorithm uses the following principles:

1. A bat uses echolocation for distance estimation and "knows" the difference between the food/prey and an obstacle

2. Bats search for their prey by flying randomly with a velocity of v_i, variable wavelength λ, fixed frequency f_{min} and loudness A_0. Depending on the proximity to the prey. bats adjust the wavelength of the emitted impulse and its level of emission in $r \in [0,1]$.

3. While the loudness can be changed by different means we assume that the loudness vary from big positive value A_0 to minimum constant A_{min}. In addition to these simplified principles, lets use the next approximations: frequency f from the segment $[fmin, fmax]$ corresponds to the wavelength segment $[\lambda_{min}, \lambda_{max}]$. For instance, a frequency segment $[20KHz, 500KHz]$ corresponds to wavelength segment $[0.7mm, 17mm]$.

2.7.3 Algorithm

```
BEGIN
f(x) is the objective function
Population of bat is initialized, xi (i = 1, 2,
    ..., n) and vi
Pulse frequency fi at xi is defined
The loudness and pulse rates  (Ai, ri) are
   intialized
   WHILE count < number of iterations
     New solutions are generated by adjusting
        frequency, and by updating velocities
        and locations/solutions
     IF rand > ri
       Among the best a solution is selected
       A solution around the best is generated
     END
     By random flight new solution is generated
     IF rand < Ai AND f(xi) < f(x*)
       Accept new solution
       increase  ri and reduce Ai
     END
     Rank bats and find current best x*
   END
   Post process results and visualize it
```

2.8 ARTIFICIAL BEE ALGORITHM

The aim of a bee swarm is to find the area of a field with the highest density of flowers [27]. Without any knowledge about a field bees begin the search of flowers from random positions with random velocity vectors. Each bee can remember positions where the maximal quantity of flowers was found and know where other bees found the maximum density of flowers. When a bee chooses between the place where it found the maximum quantity of flowers and the place which was reported by others, the bee rushes in direction between these two points and decides between personal memory and social reflex. On its way the bee can find a place with more high concentration of flowers than were found

previously. In the future this place can be marked as the one with the highest concentration of flowers found by a swarm. After that the whole swarm will rush in the direction of this place, remembering though their own observations. Thus, bees research a field by flying to places with the highest concentration of flowers. They also continuously compare places they flew over with previously found ones in order to found the absolute maxim concentration of flowers. In the end, a bee ends its flight in the place with the maximum concentration of flowers. Soon the whole swarm will locate in the neighborhood of that place.

2.8.1 Mathematical model

Employed bees, onlookers and scouts are the three groups of bees in the colony. Scouts perform random search, employed bees collect previously found food and the dances of employed bees are watched by onlookers. Onlookers also chooses the food sources depending on the dances. Onlookers and scouts are called non-working bees. Communication between bees is based on dances. Before a bee starts to collect food it watches dances of other bees. A dance is the way bees describe where food is.

Working and non-working honey bees scan for rich nourishment sources close to their hive. A working honey bee keeps the data about a nourishment source and offer it with spectators. Working honey bees whose solution can't be improved after a positive number of endeavors become scouts and their answers are not utilized after that. The quantity of sustenance sources speaks to the quantity of solutions in the populace. Conceivable solution are spoken to by the situation of a nourishment source. The nature of the solution is spoken to by the nectar measure of a sustenance source.

2.8.2 Algorithm

```
BEGIN
The population is initialized
For the initial interation current best agent
  is found
```

```
Calculate the no of scouts, onlookers and
   employed bees
SET global best to current best
FOR iterator < iteration
  fitness for each agent is evaluated
  Fitness is sorted in ascending order and find
     the best agent
  Select agents (a to c) from a list of best
    agents
  Bees flying to the best solution are created
  Current best agent is evaluated
  IF fitness value of current best < fitness
    value of global best
    global best = current best
  END
END
Global best is saved
```

2.9 FIREFLY ALGORITHM

Most species of fireflies are able to glow producing short flashes. It is considered that the main function of flashes are to attract fireflies of the opposite sex and potential preys. Besides, a signal flash can communicate to a predator that a firefly has a bitter taste. [28]

2.9.1 Mathematical model

The Firefly Algorithm is based on two important things: the change in light intensity and attractiveness. The brightness of a firefly (connected with the objective function) defines its attractiveness. The algorithm utilizes the following firefly behaviour model:

1. All fireflies are able to attract each other independently of their gender;

2. A firefly attractiveness for other individuals is proportional to its brightness.

3. Less attractive fireflies move in the direction of the most attractive one.

4. As the distance between two fireflies increases, the visible brightness of the given firefly for the other decreases.

5. If a firefly finds no other firefly brighter, then its motion is random.

2.9.2 Algorithm

```
f(x) is the objective function
  Initialize the population of fireflies xi(i =
     1, 2, ... , n)
  Define light absorption coefficient (gamma)
  WHILE count < Generations
    FOR i < no of fireflies
      FOR j < i
      Light intensity (Ii) at xi is determined
        by f(xi)
      IF Ii > Ij
          Move the firefly i towards j in all
            dimensions d
        ELSE
          Move firefly i randomly
        END
        Change attractiveness with distance r
          via exp[-gamma r2]
      Determine new solution and revise the
        light intensity
      END
    END
    Rank the fireflies by their light intensity
       and current best is found
  END
```

2.10 WHALE SWARM ALGORITHM

Whales live in groups. Whales produces sounds of a very wide range. The sound produced is linked to their migration, feeding and matting patterns. Moreover, a large part of sounds made by whales are ultrasound. They remains in touch with each other and finds food azimuth by ultrasound. Upon finding the food, they generate sounds to notify others about the quality and quantity of the food. That's a lot of information to process. Whales move to a proper location to find food. [29]

2.10.1 Mathematical model

The Whale Swarm Algorithm employees the following rules:

1. All the whales communicate with each other by ultrasound in the search area;
2. Whales posses the ability to compute distance to other whales
3. The fitness value of each whale determines the food quality and quantity
4. The whale having highest fitness value guide the nearby whales

2.10.2 Algorithm

```
BEGIN
  Agents are initialized
  Current best is found
  global best = current best
  FOR t < iterations
    FOR each agent
      Better and nearest are found
      IF  Exists
        Current agents is shifted in direction
           of its better and nearest
      END
      Current best is found
```

```
      IF current best better than global best
        global best = current best
      END
    END
    Global best is saved
END
```

2.11 PARTICLE SWARM OPTIMIZATION

A flock of birds is a good example of the collective behavior of animals. Flying in a big groups, they almost never collide with each other. A flock moves smoothly and is coordinated as if it is controlled by something and it's not about the leader of the flock. A flock of birds is a swarm intelligence and birds in it act according to certain rules. [30] The rules are the following:

1. Every bird tries to avoid collision with others;
2. Every bird moves in the close birds direction;
3. Birds try to move on the equal distance from each other;
4. A bird shares information with neighbours.

2.11.1 Mathematical model

In the PSO, agents are particles in the optimization task parameters space. On each iteration particles have some position and a velocity vector. For each position of a particle the corresponding objective function value is calculated and on the basis of that value a particle changes its position and velocity according to certain rules. The pso is a stochastic optimization method. It doesn't update existing populations but works with one static population which members steadily improve as they receive more information about the search space.

2.11.2 Algorithm

```
BEGIN
  Agents are initialized
  Current best is found
  Global best = current best
  FOR i < iterations

    Particle velocity is calculated
    Particles velocity is changed
    Particles positions is updated
    New agent is selected according to the
       selection strategy
    IF current best better than global best
      Global best = current best
    END
  END
  Global best is saved
END
```

2.12 CHICKEN SWARM OPTIMIZATION

The chicken swarm consist of a rooster and many hens and chicks. In different hierarchical order there exist competition among chickens. Different chicken follows different moving pattern. Domestic chickens like to live in flocks. Chickens shares a lot of informations related to nesting, food discovery, mating, danger etc through different sounds. Trial and error is not the only source of learning they also learn from previous experiences and from others. It's obvious that a preponderant one would dominate the weak. The head roosters are surrounded by the more dominant hens. Also, dominant hens surrounds more submissive hens. The social order would disrupt by removing or adding chickens from an existing group until a specific order is established.[31]

2.12.1 Mathematical model

In the CHSO algorithm the chickens' behaviours are described by the following rules:

1. There are many groups, with each group consisting of a rooster (dominating), couples of hens and chicks.

2. Fitness value of the chickens determine the division of chickens into several groups and determination of the identity of the chickens, here, it is rooster, hens and chicks. The head rooster of the group is the one having best fitness value in many cases. Whereas, the worst performing chicken would be considered as chick. Rest would be considered as hens. Hens are randomly selected to live in a group. Similarly, the mother to child relationship is determined randomly.

3. Some constant relationship in a group would be their hierarchical order, mother to child and dominance relationship. They would be updated after many time steps.

4. For finding food chickens would follow their rooster. Chickens would also have to protect their food being eaten by others. There will also be a possibility of food stealing by other chickens. Whereas, the chicks would restrict their search area around their mother. It's obvious that the dominant would be having advantage here.

2.12.2 Algorithm

```
BEGIN
  N chickens are initialized with their
    parameters
  Calculate fitness function values
  Find current best
  Set global best = current best
  FOR i = 0 : number of iterations
    IF(i % G == 0)
      Hierarchy is made according to the
        fitness value of chickens
      Relationship (chicks to hens) is
        determined in a group after dividing
        the swam into groups
```

```
    END IF
    FOR j = 1 : N
      IF j == rooster
        Solution and location are updated
      END IF
      IF j == hen
        Solution and location are updated
      END IF
      IF i == chick
        Solution and location are updated
      END IF
      Current best is evaluated
      IF current best better than global best
        SET global best to current best
      END IF
  END FOR
END
```

2.13 SOCIAL SPIDER ALGORITHM

The Social Spider Algorithm mimics the social spiders colony behaviour. These spiders form colonies which allow them to remain together on a communal network.

A social spider settlement comprises of two fundamental segments: its communal network and its members. All individuals are separated into two distinct groups: females and males. [32]

2.13.1 Mathematical model

This calculation depends on the presumption that the whole search space is a common web. In this web all the social-spiders communicate to one another. Every spiders position in the public web speak to an solution inside the search space. Weights are determined to every spider as per the fitness estimation of the solution. There two search agents here, male and female. Every individual is led by a lot of various transformative administrators relying upon sexual orientation which copy distinc-

tive helpful practices that are normally accepted inside the settlement. Social-spiders have profoundly female-one-sided populaces. The calculation begins by characterizing the quantity of female and male spiders that will be portrayed as people in the search space.

2.13.2 Algorithm

```
BEGIN
  Create the population of spiders
  Initialize target vibration for each spider
  FOR i = 0 : number of iterations
    FOR each spider in population
      Evaluate the fitness values of a spider
      Generate a vibration at the position of
        the spider
    END FOR
    FOR each spider in population
      Calculate the intensity of the vibrations
         generated bu other spiders
      Select the strongest vibration from all
        vibrations
      IF the intensity of the strongest
        vibration is larger than target
        vibration
        target vibration = strongest vibration
      END IF
      Perform a random walk towards target
        vibration
      Generate a random number rn from [0,1)
      IF rn < pj
        Assign a random position to the spider
      END IF
      Attenuate the intensity of target
        vibration
    END FOR
```

```
  END FOR
  Save the best solution
END
```

2.14 CAT ALGORITHM (CAT SWARM OPTIMIZATION)

Cat Algorithm mirrors the two parts of cat behaviour: seeking mode and tracking mode. A cat can be in either seeking mode or tracing mode, while comprising of M dimensional position, having speed of each measurement and a fitness esteem. The last solution would be the best position in one of the cats. [33]

2.14.1 Mathematical model

Seeking mode speak to the circumstance of the car which could be resting, glancing around or looking for the following position to move in.

In seeking mode, the four fundamental components are characterized:

1. Seeking Memory Pool (SMP)
2. Seeking Range of the selected Dimension (SRD)
3. Counts of Dimension to Change (CDC)
4. Self-Position Considering (SPC)

For the seeking mode the following algorithm is proposed:

```
FOR each cat-agent
  Create j = SMP copies
  IF SPC is true
    j = SMP - 1
    Save copies
  END IF
END FOR
FOR each copy
  Randomly add (or subtract) SRD
END FOR
FOR each copy
```

```
  Calculate fitness function value FSi
END FOR
IF all values of the fitness function are not
   equal to each other
  Calculate Pi
END IF
IF FSi are equal
  Pi = 1
END IF
FSb = FSmin
Replace cat-agent with its copy
```

Tracing mode is the second mode of a cat. In this mode the cat tracks down and attacks its prey. In tracing mode the algorithm works as follows:

```
Calculate new velocity vector value for each
   cat
Calculate new position of a cat
```

2.14.2 Algorithm

```
Initialize n cats in the domain D randomly (
   Initially each cat has zero velocity vector)
Generate a flag for each cat
FOR number of iterations
  Calculate Pbest
  Move each cat considering its flag:
    IF flag = 0
      Perform seeking mode
    ELSE
      Perform tracing mode
    END IF
  Redistribute the flags
END FOR
```

2.15 GRAVITATIONAL SEARCH ALGORITHM

The Gravitational Search Algorithm is based on the laws of gravitation and mass interaction. Basically, this algorithm is similar to Particle Swarm Optimization (PSO), since they are both based on the development of a multi-agent system. [**?**]

2.15.1 Mathematical model

GSA operates with two laws:

1. *law of gravitation*: every particle pulls in different particles and force of gravity between two particles is directly proportional to the product of their masses and inversely corresponding to the separation between them (one should pay attention to the fact that, unlike the law of universal gravitation, we don't use the square of the distance, as it results in better results of the algorithm).

$$F_1 = F_2 = G * (m_1 * m_2) * r^{-2}$$

2. *law of motion*: the present speed of any particle is equivalent to the total of the part of the speed at the past moment of time and to the adjustment in speed which is equivalent to the force the framework influences the particle with divided by the inertial mass of the particle.

2.15.2 Algorithm

```
1. Generate the system randomly;
2. Determine the fitness of each particle;
3. Update the value of the gravitational
   constant, masses and the best and the worst
   particle values;
4. Calculate the resultant force in different
   directions;
5. Calculate accelerations and velocities;
6. Update particles' positions;
```

7. Repeat steps 2 to 6 until the stop condition is reached.

CHAPTER 3: ZERO INITIALIZATION OF MODIFIED GATED RECURRENT ENCODER DECODER NETWORK

Recurrent Neural Network (RNN) are very powerful and is used for timeseries forecasting in various ways. But the problem with such type of network is that we always need a FNN as a last layer and no specific technique for parameter initialization exist for such layer. When we look at some of the works done in this field such as Xavier[35], He[36] and identity matrix [37], we see significant results with Rectified Linear Unit (ReLu) or its variants on a deep neural network. Other recent work include Layer-sequential unit-variance (LSUV)[38] initialization which also works great with ReLu family and deep network. All such type of initialization were designed so that the input signal could traverse deep in the network. But using these techniques on a single layer network makes no sense. But we could still see these type of techniques being used on a single FNN. Thus, we propose a ZI for single FNN and tested it against various initialization techniques and implemented it on prize winning models. Results shows that ZI is way much superior than any other technique out there. We further investigate how a FNN does not die with both weights and biases initialized to zero and why it improves the accuracy.

Electricity market has changed significantly in last decade forming a restructured market[50] where the prediction has become difficult due to emerging technologies. Short term load forecasting is particularly important for power system security and electricity cost. Mainly two types of forecasting is done, one is point forecasting and other is probability forecasting. If forecasting is done inaccurately it would lead to increased operating cost due to allocation of insufficient reserve capacity and use of expensive peak load units. Also, when talking about spot price and electrical energy trading in open market, electricity load fore-

casting plays a major role to purchase the electricity at minimum cost. Point forecasting has emerged as the dominating technique for prediction of load with the major models being exponential smoothing[44] and regression[45, 50] in the field of statistical models.

When talking about artificial neural network[51], models with variant of neural network have shown significant results in this field such as Improved Neural Network[43, 48], Support Vector Machine [46] and Fuzzy System [47, 49]. There are two major problems associated with artificial neural network, namely, "overfitting" in which the model fails to train for the underlying relationship and the other is "dimentionality" in which the models complexity increases exponentially as the number of input dimension increases.

The focus of this paper is to use a GRU encoder-decoder model to maximize the conditional probability of the target sequence given the input sequence. This model is mainly used in Neural Machine Translation where the input sequence length is variable in nature along with the output sequence length. In the decoder of the model, the fully connected dense neural network was initialized with the defamed Zero Initialization technique. We further investigate how initialization of both weights and biases to zero could be possible and why we didn't end up with dead neurons. It was shown that with only changing the initialization technique of this output layer we could get high level of accuracy in comparison to other initialization techniques.

For testing of our models we have used electricity load of New South Wales (NSW). The timeseries is half hourly distributed and we have taken 48 hours as our forecasting horizon. We have two models and used various initialization techniques to compare our results. The results show that convergence speed and variance shown when the model initialized with ZI were superior to when initialized with other techniques. The mean absolute percentage error(MAPE) in load forecasting with ZI was around 0.94% while with Xavier Uniform was around 1.06%. Other techniques showed higher error and variance.

3.1 EXPERIMENT SETUP

3.1.1 Dataset used

For our models we have used electricity load of New South Wales (NSW)[39] of Australian Energy Market. The feature used from the dataset includes only total demands and its time. We have used historical electricity load data of year 2015,2016,2017.

3.1.2 Data pre-processing and feature engineering

Raw input values was transformed by *log(1 + x)*. We first try to find any type of auto-correlation within the data. The auto-correlation plot showed strong yearly and quarterly seasonality in data and we tried to use that to increase the accuracy. First thing we did with the data was to find the annual and quarter autocorrelation of the input dataset. But due to unevenness of input interval due to leap years, monthly length differences, autocorrelation was found considering neighbouring data also to reduce noise as follows:

$$corr = 0.5 * corr(lag) + 0.25 * corr(lag - 1) + 0.25 * corr(lag + 1)$$

where, *corr(lag)* is the autocorrelation with *lag*. Two *lag*s were used here; one of 365 and other of $365.25/4$. These two values is then used as a feature. These were time independent features so it was necessary to stretch it to the timeseries length, i.e., they were added to at the end of each input batch. Next feature which we selected was day-of week. Since it could not be directly feed to the network, it was normalized as follows:

```
feature = {mon: 0, ..., sun:6}

for each feature_value in feature:
   normed = feature_value / (7 / 2*pi)
   dow = [cos(normed), sin(normed)]
```

dow from above was used as additional features. One of the important feature used was the *lagged_data*. Since data showed string quarter and

annual correlation we decided to lag the input load by $3, 6, 9, 12$ months and add them as additional features as follows:

```
timeseries_indices: time indices of
          the electricity load data
offset: in months

for each time in timeseries_indices:
    for each offset in {3, 6, 9, 12}:
        date = time - offset
        lagged_data = data(date)
```

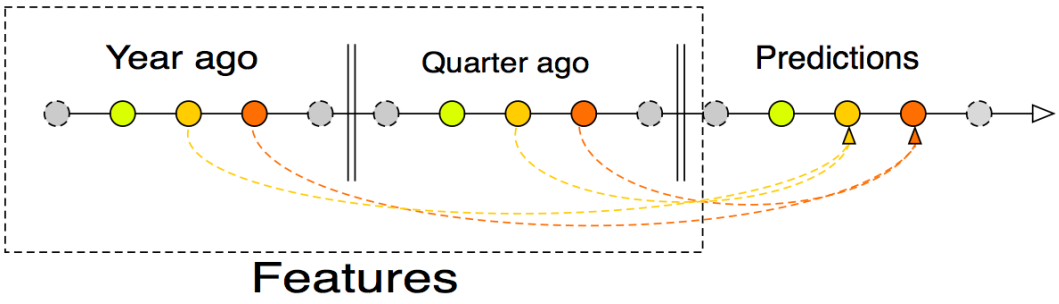
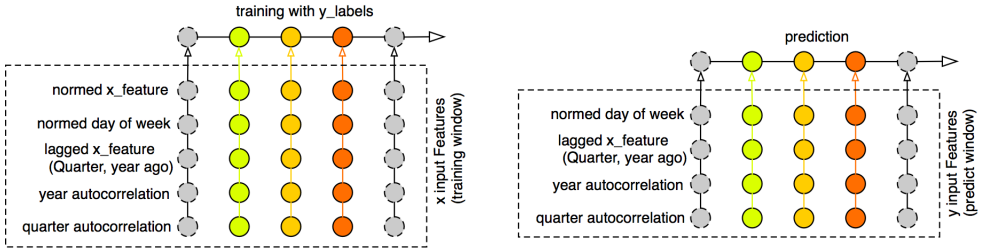


Figure 3.1: Quarter and annual feature selection for lagged_data

Important features from the past are now explicitly included in the features and hence our model need not to remember very old informations. Lagged datapoints helps in reducing the size of model and hence helps in faster training and less loss of information. This technique being simple helps in achieving high accuracy. Also, LSTM/GRU units tends to forget oldest informations when the input sequence size increases.



(a) (b)

Figure 3.2: Visualization of input features: (a) shows x input features used for training; (b) shows y input features used for prediction.

Figure 3.2 shows how features were combined as input pipeline for being feed to the model. *x_features* were splitted into batches of size *training_window* and similarly *y_features* were splitted into batches of size *predict_window*. Year and quarter autocorrleation were tiled according to the size of input features.

3.1.3 Prediction Model 1

We have done majority of the work to get the accuracy in the input pipeline so we tried to keep our first model as simple as possible. With this model we try to understand the concept of ZI and its significance. A simple RNN was selected and we used GRU [40] cells instead of LSTM. The output from the GRU layer is then connected to a fully connected (FC) dense layer. This is the layer where we will apply our initilization technique and compare the results with different initilization techniques. Figure 3.3 shows the graphical depiction of our first model.

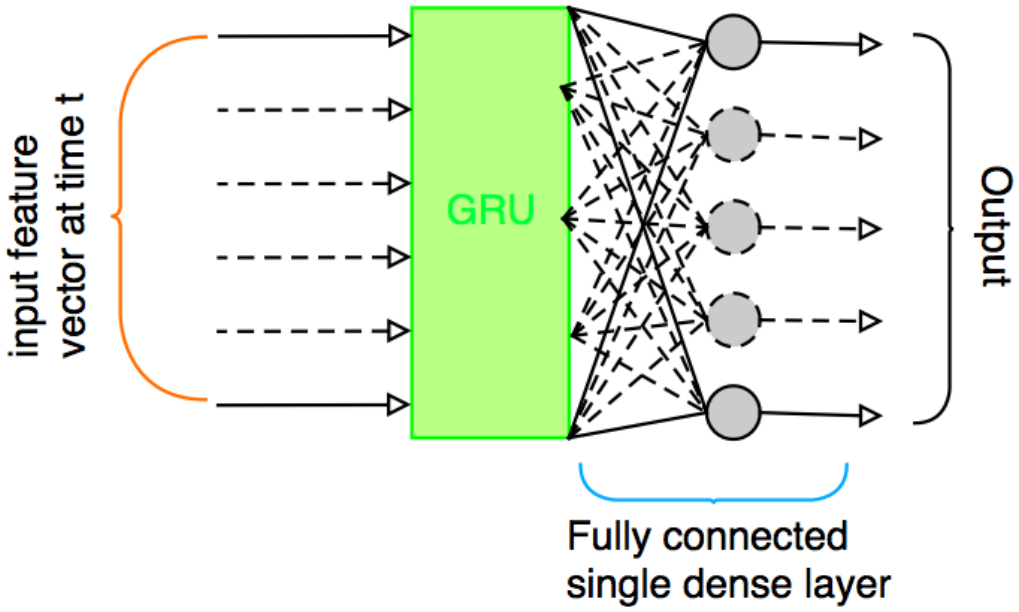


Figure 3.3: First model for electricity load forecasting with FC output layer

Analysis of ZI

Let us consider w_{jk}^{L} to be the weight connecting k^{th} neuron in the $(L-1)^{\text{th}}$ layer to the j^{th} neuron in the L^{th} layer. Here, L^{th} layer is our last FC

layer and $(L-1)^{\text{th}}$ layer is the output layer of our GRU block. For simplicity, first let us consider identity function to be our activation function. The activation of the j^{th} neuron of the last layer will be:

$$a_j^L = \sum_k (w_{jk}^{L-1} a_k^{L-1} + b_j^L)$$

We will consider quadratic cost function C as follows:

$$C = \frac{1}{2n} \sum_x abs(y(x) - a^L(x))^2 \tag{3.1}$$

where n is the size of the training dataset. Also, x is the input and $y = y(x)$ is the desired output. The weights are updated by finding tbe gradient of the cost function w.r.t. the weights as follows:

$$w_{jk}^L \leftarrow w_{jk}^L - \eta \frac{\partial C}{\partial w_{jk}^L} \tag{3.2}$$

where, η is the learining rate. From equation 3.2 it can be seen that as long as the gradient is coming out to be non zero the weights would be updated and our neuron would not die. Since, we have used qudratic cost function it's derivative by the backpropogation equation [41] can be found easily as follows:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{3.3}$$

$$\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j) \tag{3.4}$$

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1} \delta_j^L \tag{3.5}$$

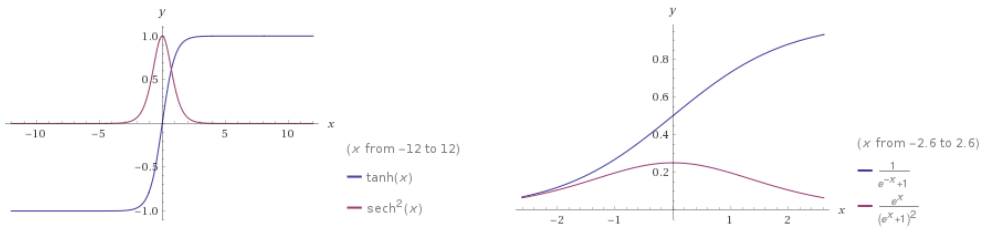
Since, our activation function was an identity, $\sigma'(z_j^L) = 1$. So, our updating gradient will be :

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1} * (a_j^L - y_j) * 1 \tag{3.6}$$

Since, our gradient came out to be non zero our neuron will keep on learning. Any activation function which have non zero derivative at zero

will keep our neuron alive. Some of the alternative activation functions which could have been used are:

$$\begin{aligned} a(z) = tanh(z) \quad &; \quad a'(z) = sech^2(z) \\ a(z) = sigmoid(z) \quad &; \quad a'(z) = a(z)(1-a(z)) \end{aligned}$$



(a) (b)

Figure 3.4: Plot of different activation function with their derivatives: (a) $tanh(x)$ and $sech^2(x)$; (b) $sigmoid(x)$ and $sigmoid(x)(1-sigmoid(x))$.

From figure 3.4 it's clear that the derivative of the activation functions are well defined at zero and are non-zero.

At the begining of the training all the weights and biases will be initialized to zero, so , $a_j^L = 0$. But $a_k^{L-1} \neq 0$ since it is the output from the GRU block.So, it can be assumed that each neuron output coming from GRU will be different. With these assumptions the gradient at the begining will be

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1} * (-y_j) \tag{3.7}$$

Because of the different output coming from the GRU block, each neuron in FC layer will learn different weights with each iterations. Also, after training the weights are more tends towards zero, so it make sense to initialize all the weight to zero.

3.1.4 Prediction Model 2

A sequence to sequence (seq2seq) learning model [42] was selected to predict electricity load with high accuracy. This model was selected due to following reasons:

1. Predictions is based upon conditional probability of previous values including our past predictions.

2. This model is versatile with the type of features being injected into it, like numerical, categorical, timeseries etc.

For a given input sequence $(x_1, x_2, \ldots, x_n)$ a GRU computes the output sequence $(y_1, y_2, \ldots, y_m)$ by deciding which information to store and which information to drop. This is decided by using of a *reset* gate and an *update* gate. When the *reset* gate is close to zero the hidden unit is forced to drop previous hidden information and store the current input. Through this any information that is not relevant is dropped. The *update* gate controls how much information which be transferred from previous hidden state to the new hidden state.

The *reset* gate of the jth hidden unit is computed as

$$r^j = \sigma\left(W_r^j x^j + U_r^j h^j(t-1)\right) \tag{3.8}$$

where, W_r^j and U_r^j are the weights to be learned. x^j is the inputs to jth hidden state and $h^j(t-1)$ is the previous hidden state. σ being the sigmoid function.

The *update* gate is computed as

$$z^j = \sigma\left(W_z^j x^j + U_z^j h^j(t-1)\right) \tag{3.9}$$

where, the notations being similar to the *reset* gate. The activation of the hidden state is computed as

$$h^j(t) = z^j h^j(t-1) + (1-z)^j \hat{h}^j(t)) \tag{3.10}$$

where,

$$\hat{h}^j(t) = \phi\left(W_h^j x^j + U_h^j (r^j \odot h^j(t-1))\right) \tag{3.11}$$

where, ϕ being the $tanh$ activation function and $\odot$ is element wise product.

The output of GRU is then calculated as

$$y(t) = W^L h(t) \tag{3.12}$$

With these GRU units the model 2 was constructed with the aim to calculate the conditional probability $p(y_1, \ldots, y_m | x_1, \ldots, y_n)$ where, x

and y are the input and output sequences respectively. The length of input and output sequences, n and m may or may not be equal. Since the length of input and output sequences may be different an Encoder is used in this model that reduces the input sequence into a fixed dimensional vector c. The hidden state of this encoder h is than transferred to the decoder along with the last input of the input sequence x. The decoder utilizes the hidden state of the encoder and previous output to estimate the conditional probability as

$$p((y_1,\ldots,p_m)|(x_1,\ldots,x_n)) = \prod_{t=1}^{m} p(y_t|(h,y_1,\ldots,y_{t-1})) \tag{3.13}$$

Figure 3.5 depicts how the second model was constructed.

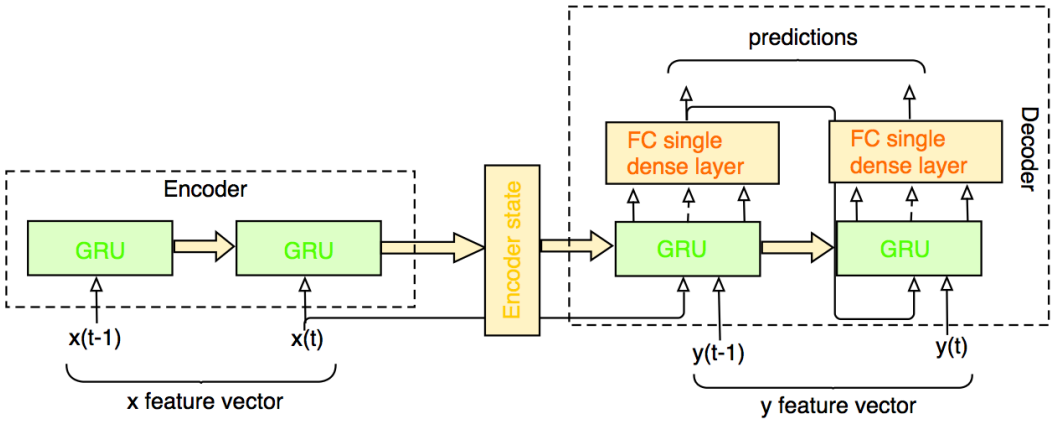


Figure 3.5: Second model for electricity load forecasting with FC output layer

Losses and regularization

Smoothed Differential Symmetric Mean Absolute Percentage Error (SSMAPE) was used as a loss function in our model which is a variant of SMAPE. Mean Absolute Error (MAE) could have also been used on $log(1+x)$. The reason for using these two was that their behaviour is well defined when truth value was close to zero and predicted value moves around zero. Figure 3.6(a) shows this problem graphically. The final predicted values were rounded to their nearest integer value and negative values were clipped to zero as can been seen in figure 3.6.

$$SMAPE = \frac{1}{N}\sum \frac{2 * |F(t) - A(t)|}{|F(t)| + |A(t)|} \tag{3.14}$$

$$SSMAPE = \frac{1}{N}\sum \frac{2 * |F(t) - A(t)|}{Max(|F(t)| + |A(t)| + \varepsilon, 0.5 + \varepsilon)} \tag{3.15}$$

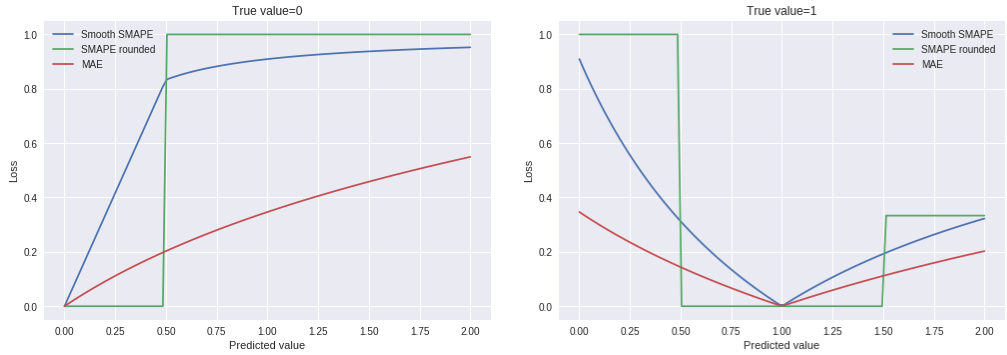
$$MAE = \frac{1}{N}\sum |F(t) - A(t)| \tag{3.16}$$

where $F(t)$ is the forecasted value and $A(t)$ is the actual value. ε is the smoothness factor in SSMAPE.

Additional cost term for RNN were added to both encoder and decoder which adds squared magnitude of coefficient as penalty.

$$L2 = \frac{\beta}{2}\sum R(t)^2 \tag{3.17}$$

where, $R(t)$ is the RNN output and β is the hyperparameter controlling the amount of regularization.



(a) (b)

Figure 3.6: Plot of different loss functions: (a) when True value = 0; (b) when True value = 1.

3.1.5 Reducing model variance

It was hard to know which training step would be best for predicting the future, so *early stopping*[41] could not be used. So, it was necessary to use some technique to reduce the variance. Averaging Stochastic Gradient Descent (ASGD) was used to reduce variance in both model 1 and 2. With ASGD the moving average of the network weights are maintained during training. These averaged weights are used during predictions in-

stead of the original ones.It has to be noted that if ASGD was used from the starting of training, the training would be very slow, so it was necessary to apply ASGD after certain epochs have been passed. ASGD was used only in FC layer for averaging of weights.

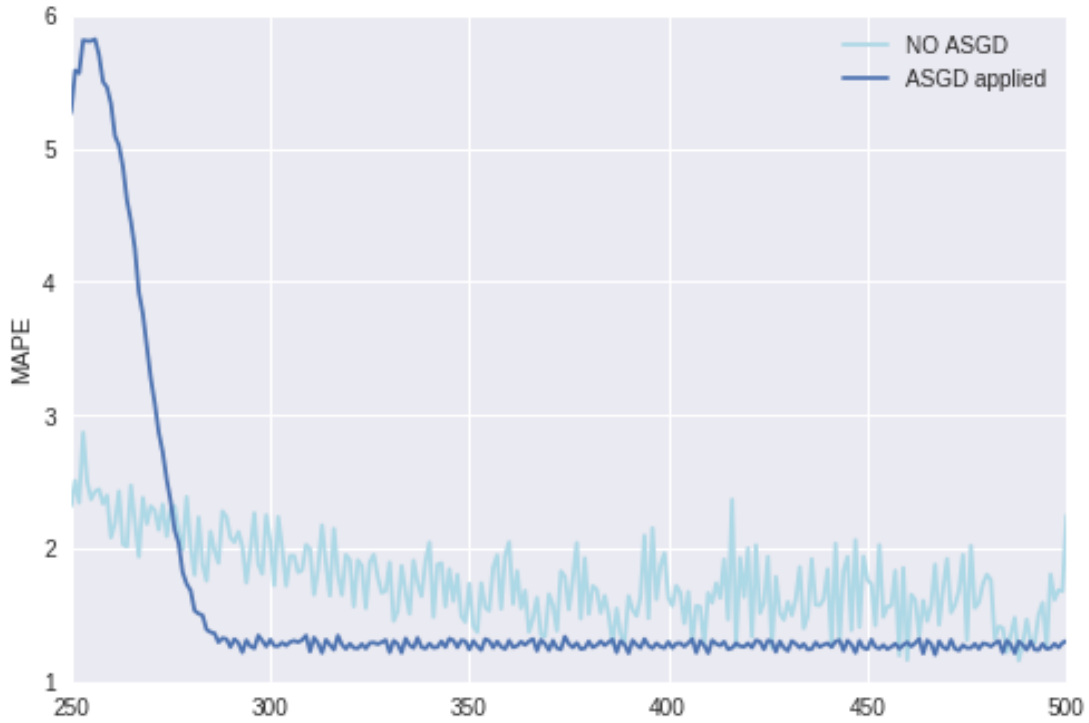


Figure 3.7: Variance reduction applied to the weights of FC layer of model 1

3.1.6 Training and validation

Five fold cross validation was used for the comparison purpose. The training and validation dataset was repeated within each epochs so as to reduce the number of epochs and improve the accuracy. The repetition of dataset was achieved as follows:

```
data_train = data_train.repeat(n_repeat)

for epoch in epochs:
  train(data_train)
```

CHAPTER 4: RESULTS AND CONCLUSION

4.1 SIMULATION RESULTS

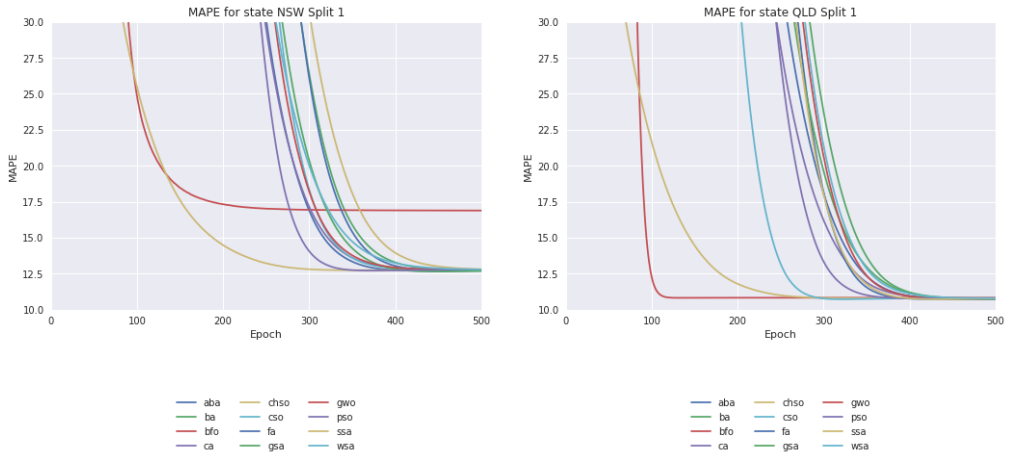
For swarm intelligence the simulation was done by splitting the dataset into two types, namely timeseries split and 5 fold cross validation. Two types of weight initialization were compared; one was using random Gaussian variable with zero mean and $1/\sqrt{n_{in}}$ standard deviation (from now onwards calling it "default") and the other was using swarm intelligence algorithms. The following parameters were common in all the simulation:

ANN Cost function	Quadratic Cost
ANN Activation function	tanh
Regularization	L1
Lambda	2
Hidden neurons	10

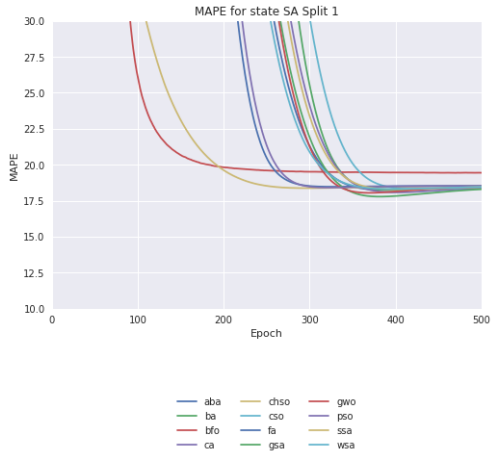
Convergence of MAPE with epochs for each states and each types of splitting and initialization is shown in figure 4.1.

The initialization started with Gaussian distribution and it was found that the Cuckoo Search initialization was superior to it. Based on that results, the initialization was tested further with other swarm intelligence algorithms. The results shown in figure 4.1 shows that the Chicken Swarm Optimization (chso) and Bacterial Foraging Optimization (BFO) gave comparable results to each other and were way much superior to Gaussian distribution. If both figure 2.1 and 4.1 are compared to each other it can be found that with the Gaussian initialization the solution saturated at around 1100 epochs while with the swarm intelligence it saturated at around 200 epochs. Same simulation when done with 5 fold cross validation showed same result with a considerably reduction in number of epochs.

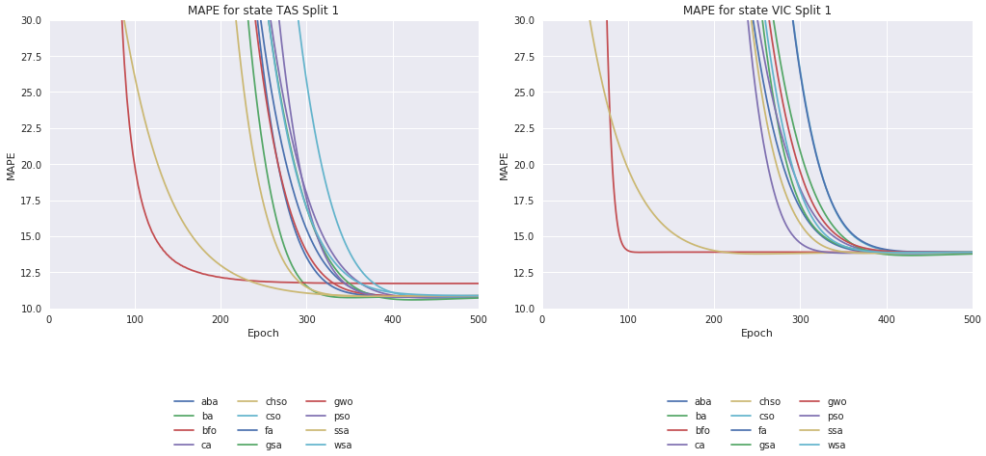
For ZI, two variants of Xavier initialization were used in testing the



(a) (b)



(c)



(d) (e)

Figure 4.1: Graphs showing convergence during training using swarm intelligence for states: (a) NSW; (b) QLD; (c) SA; (d) TAS; and, (e) VIC.

Table 4.1: Model 2 with Xa norm initialization

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	STDEV	Mean
NSW	6.5929	5.9959	5.0434	5.5507	5.2175	0.6263	5.6801
QLD	4.5606	3.7321	3.2218	5.3963	3.9278	0.8376	4.1677
SA	8.6941	10.4973	9.1520	10.8591	10.5814	0.9667	9.9568
TAS	4.9395	4.8040	8.4487	5.8538	5.5170	1.4805	5.9126
VIC	7.3316	7.0493	5.6297	10.0888	6.0339	1.7469	7.2267

Table 4.2: Model 2 with He initialization

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	STDEV	Mean
NSW	4.6513	5.6656	5.0018	5.7170	4.5206	0.5582	5.1113
QLD	4.8662	3.8662	3.5139	5.8110	3.9960	0.9275	4.4106
SA	8.4525	10.8284	9.2735	11.3734	10.6170	1.2058	10.1090
TAS	5.0431	5.3278	9.8243	5.7468	4.9529	2.0611	6.1790
VIC	7.4215	6.7470	5.4655	10.0543	5.2911	1.9265	6.9959

Table 4.3: Model 2 with Identity initialization

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	STDEV	Mean
NSW	5.3960	6.0436	5.2154	5.9269	5.2304	0.3946	5.5625
QLD	6.9085	3.8863	3.3958	5.3821	4.7401	1.3762	4.8626
SA	9.8609	11.7342	9.6232	11.2667	11.7405	1.0286	10.8451
TAS	5.5648	5.0953	9.7718	6.6790	5.3306	1.9334	6.4883
VIC	6.5635	7.5402	5.4254	8.7992	6.3995	1.2792	6.9455

Table 4.4: Model 2 with ZI initialization

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	STDEV	Mean
NSW	3.6601	4.1024	3.6046	4.2836	3.6129	0.3179	3.8527
QLD	3.8734	3.0213	3.5998	4.5921	3.5880	0.5708	3.7349
SA	7.2213	8.0867	6.9555	9.4038	7.8146	0.9561	7.8964
TAS	5.3289	4.2857	7.9611	6.3351	4.4796	1.5117	5.6781
VIC	6.2582	5.8454	4.7763	6.3946	4.6321	0.8274	5.5813

Table 4.5: Model 2 with Xa uniform initialization

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	STDEV	Mean
NSW	5.2228	5.3811	5.4969	5.6936	5.0978	0.2326	5.3785
QLD	6.2597	4.6270	3.0443	5.8337	4.0537	1.3098	4.7637
SA	8.7608	11.9020	9.4006	9.9402	9.3062	1.2147	9.8620
TAS	5.0635	5.0450	8.9477	5.9480	5.0639	1.6849	6.0136
VIC	7.1276	7.1873	5.2693	7.4563	6.0190	0.9306	6.6119

Table 4.6: Model 2 with all the initializations

	NSW	QLD	SA	TAS	VIC
Xa_norm	5.6801	4.1677	9.9568	5.9126	7.2267
He	5.1113	4.4106	10.1090	6.1790	6.9959
Identity	5.5625	4.8626	10.8451	6.4883	6.9455
ZI	3.8527	3.7349	7.8964	5.6781	5.5813
Xa_uniform	5.3785	4.7637	9.8620	6.0136	6.6119

models. The first one draws samples from normal distribution and the second one draws from uniform distribution within the $[-limit, limit]$. Where, $limit$ is defined for the normal distribution as,

$$limit = \sqrt{\frac{2}{in + out}} \tag{4.1}$$

and for uniform distribution as,

$$limit = \sqrt{\frac{6}{in + out}} \tag{4.2}$$

where, *in* and *out* are the number of input and output units in the weight vector [35]. Xavier uniform initializer is also called Glorot uniform initializer.

The second initializer called He, draws samples from a truncated normal distribution within $[-limit, limit]$ where $limit$ is,

$$limit = \sqrt{\frac{2}{in}} \tag{4.3}$$

its variant draws samples from uniform distribution within $[-limit, limit]$

$$limit = \sqrt{\frac{6}{in}} \tag{4.4}$$

The Identity initializer used is a simple one which utilizes an identity matrix for initialization.

All the initialization techniques where used for initialization of model having single layer output network and the results are shown in table 4.1 to 4.5. From table 4.6 one can see that the best performance was given by the model when the model was initialized with zero. From table 4.6 the average MAPE we got with ZI were around 3.8527, 3.7349, 7.8964,

5.6781 and 5.5813 for the five states which were superior to any other MAPE given by other initialization techniques.

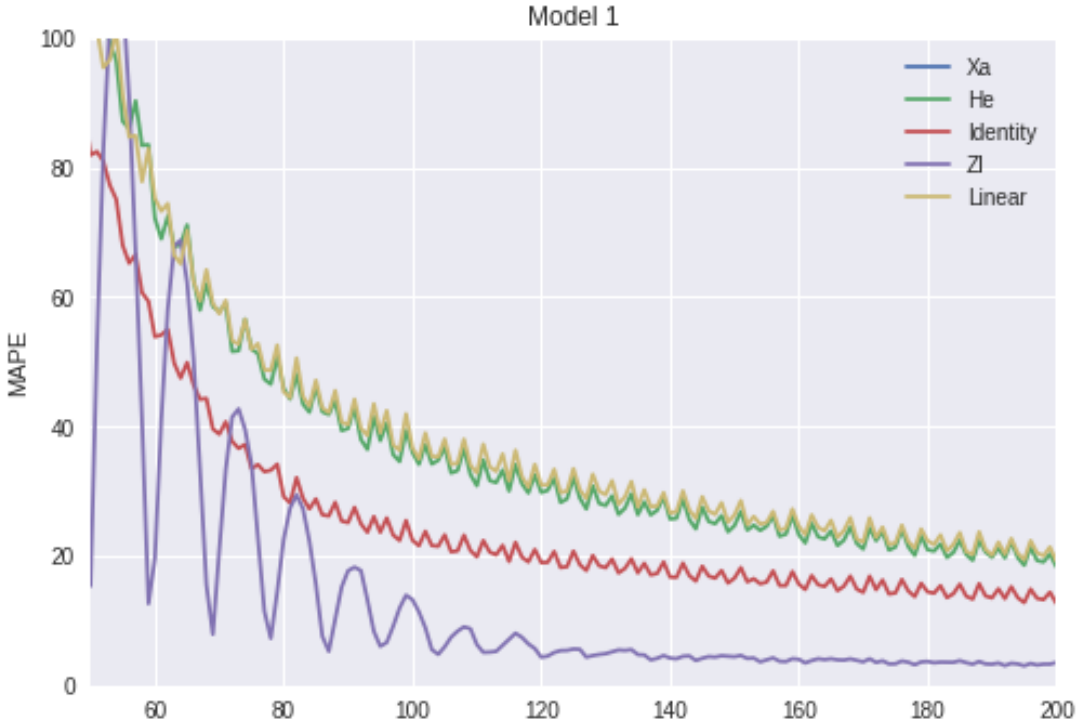


Figure 4.2: Training steps from model 1 using various initialization techniques

To see how well the convergence is taking place during training, the plot during convergence was drawn in figure 4.2. Figure 4.2 was drawn during training of first model. From there one could see that the model is converging quite well with the ZI technique.

The plot of error in output of each 48 hour of testing dataset is shown in figure 4.3. Figure 4.4 shows the output of the model 2 against the actual output. From figure 4.3 and 4.4 one can see that ZI showed best performance.

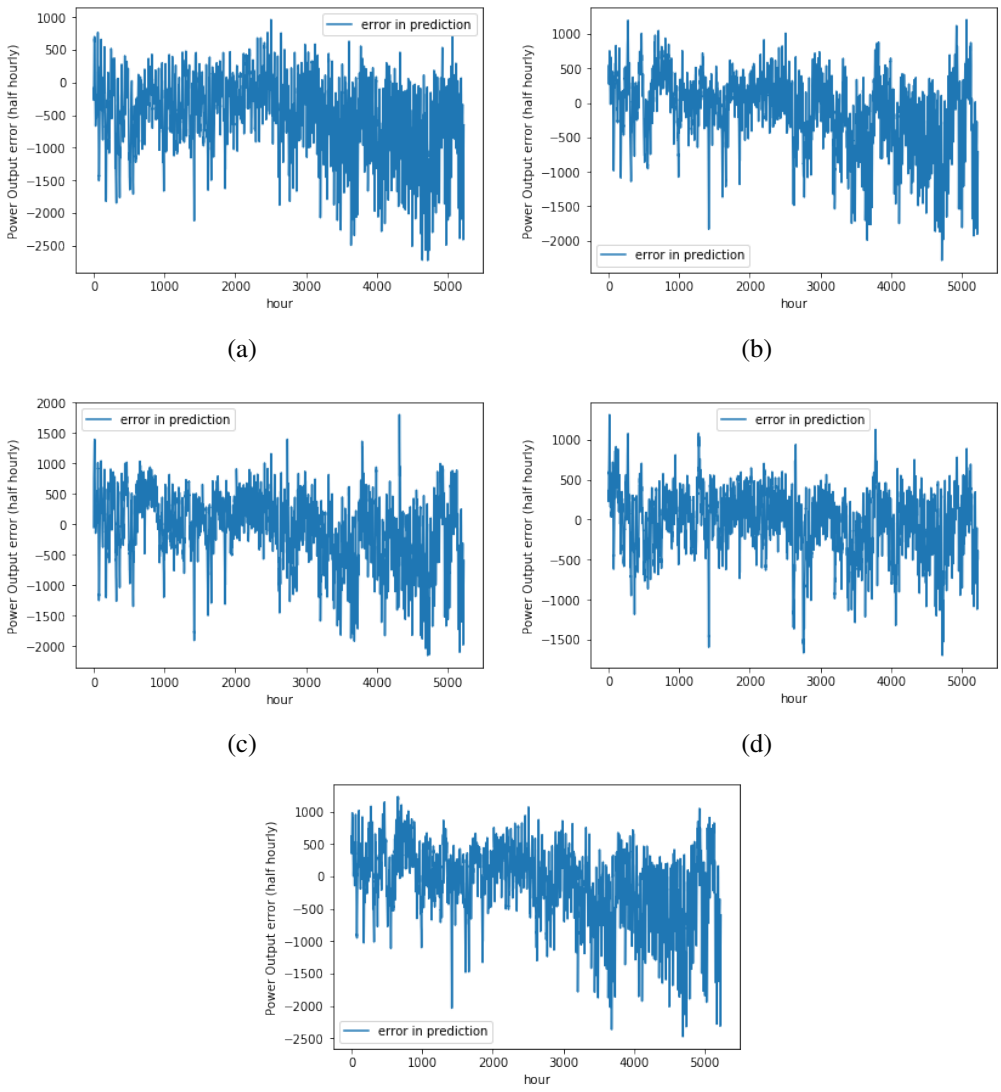


Figure 4.3: Visualization of error on validation set of state NSW (Fold 1) : (a) shows error with XaNorm initialization; (b) shows error with He initialization; (c) shows error with Identity initialization ; (d) shows error with ZI initialization; and, (e) shows error with XaUniform initialization ;

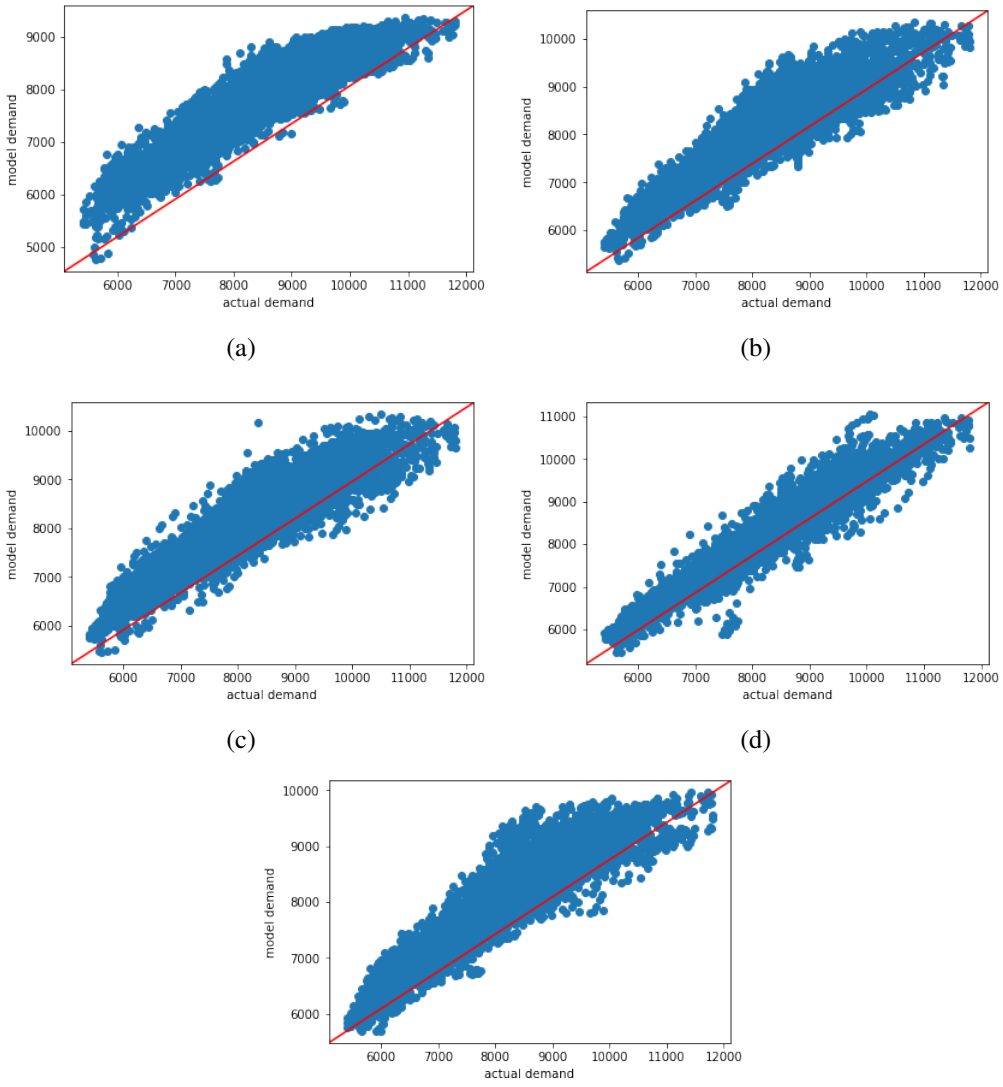


Figure 4.4: Visualization of output vs actual on validation set of state NSW (Fold 1) : (a)with XaNorm initialization; (b)with He initialization; (c) with Identity initialization ; (d)with ZI initialization; and, (e) with XaUniform initialization ;

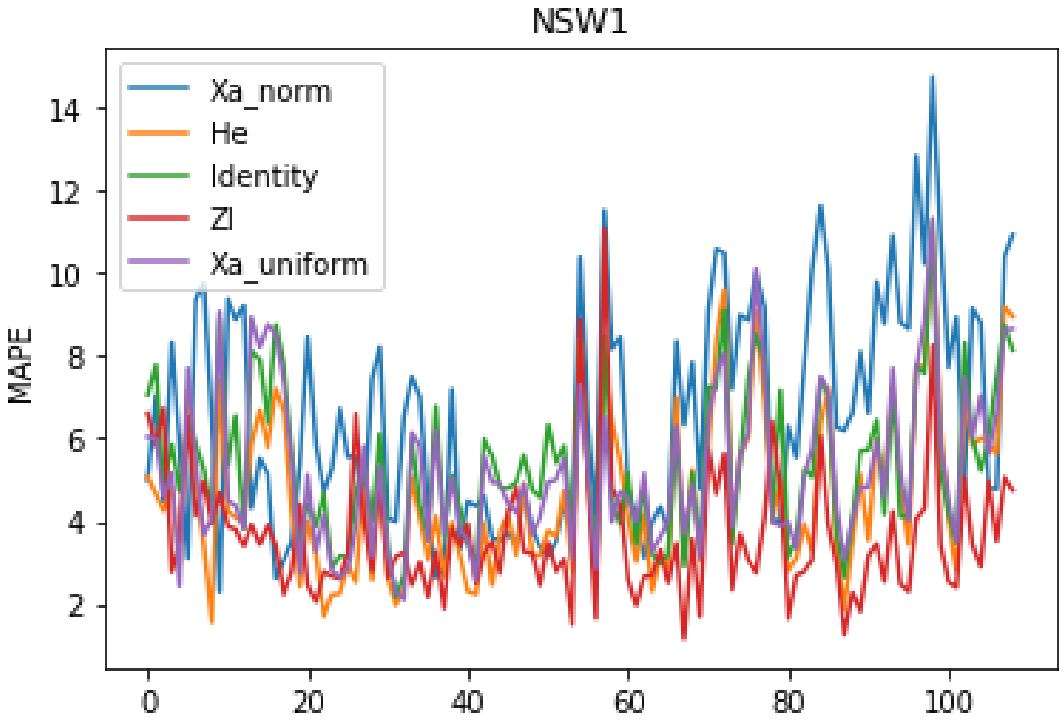


Figure 4.5: MAPE of model 2 (Fold 1) with all initialization techniques.

To see how well technique performed, MAPE from model 2 with all initialization techniques were plotted in figure 4.5. From there it can be seen that Model 2 used with ZI showed minimum MAPE in comparison to other techniques.

4.2 CONCLUSIONS

It was found that dataset splitting using timeseries and 5 fold cross validation doesn't make much of a difference in reducing the error. When talking about the method of initialization, it was found that CSO initialization was successful in reducing the number of epochs and hence helps in faster convergence. Based on that conclusion best swarm intelligence optimizer was searched and it was found that chicken swarm and bacterial foraging were way much faster in converging to the solution. It should be noted that the number of hidden layers used in the simulation was 10 which is less than used in other researches. The overall aim of this simulation was to use less computing resources for faster convergence of the solution. Hence, an ANN with one hidden layer with 10 neurons were used in the simulation.

After searching for the best swarm intelligence, use of zero initialization technique for single layer output of a complex network was introduced. The proposed model was able to reduce the variance and also increase the speed of training. This technique was evaluated with other modern initialization techniques and model and it was shown that it was superior to other techniques. The model used was a seq2seq network since the output is based upon the conditional probability of previous output and input. Also, seq2seq network could be extended for other timeseries forecasting such as webpage views on a daily basis and stock price predictions.

Before feeding the model with the input pipeline, some preprocessing of data such as autocorrelation within the data was done. The dataset showed strong annual and quarter correlations which was fed into the input pipelines. It was also shown why some activation functions can't be used with ZI.

BIBLIOGRAPHY

[1] Park, Dong C. et al. “Electric load forecasting using an artificial neural network”. IEEE transactions on Power Systems, Vol. 6, Issue 2, 1991, pp 442-449.

[2] Hecht-Nielsen, Robert. “Theory of the backpropagation neural network”. Neural networks for perception, 1992, pp 65-93.

[3] Horn, Roger A. “The hadamard product”. Proc. Symp. Appl. Math, Vol. 40, 1990.

[4] Tosun, Ömür. “Cuckoo search algorithm”. Encyclopedia of Business Analytics and Optimization. IGI Global, 2014. pp 558-564.

[5] Bakirtzis, A. G., et al. “A neural network short term load forecasting model for the Greek power system”. IEEE Transactions on power systems, Vol 11, Issue 2, 1996, pp 858-863.

[6] Park, Dong C., et al. “Electric load forecasting using an artificial neural network”. IEEE transactions on Power Systems, Vol 6, Issue 2, 1991, pp 442-449.

[7] Hao, Alex D. Papalexopoulos Shangyou. “An implementation of a neural network based load forecasting model for the EMS”. IEEE transactions on Power Systems, Vol 9, Issue 4, 1994.

[8] Paatero, Jukka V., and Peter D. Lund. “A model for generating household electricity load profiles”. International journal of energy research, Vol 30, Issue 5, 2006, pp 273-290.

[9] Pardo, Angel, Vicente Meneu, and Enric Valor. “Temperature and seasonality influences on Spanish electricity load”. Energy Economics, Vol 24, Issue 1, 2002, pp 55-70.

[10] Hong, Tao, and Shu Fan. “Probabilistic electric load forecasting: A tutorial review”. International Journal of Forecasting, Vol 32, Issue 3, 2016, pp 914-938.

[11] Soares, Lacir J., and Marcelo C. Medeiros. “Modeling and forecasting short-term electricity load: A comparison of methods with an application to Brazilian data”. International Journal of Forecasting Vol 24, Issue 4, 2008, pp 630-644.

[12] Amaral, Luiz Felipe, Reinaldo Castro Souza, and Maxwell Stevenson. “A smooth transition periodic autoregressive (STPAR) model for short-term load forecasting”. International Journal of Forecasting, Vol 24, Issue 4, 2008, pp 603-615.

[13] Antoniadis, Anestis, et al. “A prediction interval for a function-valued forecast model: Application to load forecasting”. International Journal of Forecasting, Vol 32, Issue 3, 2016, pp 939-947.

[14] Wang, Pu, Bidong Liu, and Tao Hong. “Electric load forecasting with recency effect: A big data approach”. International Journal of Forecasting, Vol 32, Issue 3, 2016, pp 585-597.

[15] Berk, K., A. Hoffmann, and A. Müller. “Probabilistic forecasting of industrial electricity load with regime switching behavior”. International Journal of Forecasting, Vol 34, Issue 2, 2018, pp 147-162.

[16] Quan, Hao, Dipti Srinivasan, and Abbas Khosravi. “Short-term load and wind power forecasting using neural network-based prediction intervals”. IEEE transactions on neural networks and learning systems, Vol 25, Issue 2, 2014, pp 303-315.

[17] Jurado, Sergio, et al. “Hybrid methodologies for electricity load forecasting: Entropy-based feature selection with machine learning and soft computing techniques”. Energy Vol 86, 2015, pp 276-291.

[18] Chae, Young Tae, et al. “Artificial neural network model for forecasting sub-hourly electricity usage in commercial buildings”. Energy and Buildings Vol 111, 2016, pp 184-194.

[19] Panapakidis, Ioannis P., and Athanasios S. Dagoumas. “Day-ahead electricity price forecasting via the application of artificial neural network based models”. Applied Energy Vol 172, 2016, pp 132-151.

[20] Hippert, H. S., D. W. Bunn, and R. C. Souza. “Large neural networks for electricity load forecasting: Are they overfitted?”. International Journal of forecasting Vol 21, Issue 3, 2005, pp 425-434.

[21] Dyson, Freeman. “A meeting with Enrico Fermi”. Nature Vol 427, Issue 6972, 2004, pp 297.

[22] Yang, Xin-She, and Suash Deb. "Engineering optimisation by cuckoo search." arXiv preprint arXiv:1005.2908 (2010).

[23] Tripathi, M. M., K. G. Upadhyay, and S. N. Singh. “Short-term load forecasting using generalized regression and probabilistic neural networks in the electricity market”. The Electricity Journal Vol 21, Issue 9, 2008, pp 24-34.

[24] Das, Swagatam, et al. "Bacterial foraging optimization algorithm: theoretical foundations, analysis, and applications." Foundations of Computational Intelligence Volume 3. Springer, Berlin, Heidelberg, 2009. 23-55.

[25] Emary, E., et al. "Multi-objective gray-wolf optimization for attribute reduction." Procedia Computer Science 65 (2015): 623-632.

[26] Yang, Xin-She, and Amir Hossein Gandomi. "Bat algorithm: a novel approach for global engineering optimization." Engineering Computations 29.5 (2012): 464-483.

[27] Karaboga, Dervis, and Bahriye Basturk. "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm." Journal of global optimization 39.3 (2007): 459-471.

[28] Yang, Xin-She. "Firefly algorithms for multimodal optimization." International symposium on stochastic algorithms. Springer, Berlin, Heidelberg, 2009.

[29] Mirjalili, Seyedali, and Andrew Lewis. "The whale optimization algorithm." Advances in engineering software 95 (2016): 51-67.

[30] Shi, Yuhui. "Particle swarm optimization: developments, applications and resources." Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546). Vol. 1. IEEE, 2001.

[31] Meng, Xianbing, et al. "A new bio-inspired algorithm: chicken swarm optimization." International conference in swarm intelligence. Springer, Cham, 2014.

[32] Cuevas, Erik, et al. "A swarm optimization algorithm inspired in the behavior of the social-spider." Expert Systems with Applications 40.16 (2013): 6374-6384.

[33] Chu, Shu-Chuan, Pei-Wei Tsai, and Jeng-Shyang Pan. "Cat swarm optimization." Pacific Rim international conference on artificial intelligence. Springer, Berlin, Heidelberg, 2006.

[34] Rashedi, Esmat, Hossein Nezamabadi-Pour, and Saeid Saryazdi. "GSA: a gravitational search algorithm." Information sciences 179.13 (2009): 2232-2248.

[35] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. InProceedings of the thirteenth international conference on artificial intelligence and statistics 2010 Mar 31 (pp. 249-256).

[36] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. InProceedings of the IEEE international conference on computer vision 2015 (pp. 1026-1034).

[37] Le QV, Jaitly N, Hinton GE. A simple way to initialize recurrent networks of rectified linear units. arXiv preprint arXiv:1504.00941. 2015 Apr 3.

[38] Mishkin D, Matas J. All you need is a good init. arXiv preprint arXiv:1511.06422. 2015 Nov 19.

[39] Australian Energy Market Operator, https://aemo.com.au/Electricity/National-Electricity-Market-NEM/Data-dashboard#aggregated-data

[40] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. 2014 Jun 3.

[41] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

[42] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. InAdvances in neural information processing systems 2014 (pp. 3104-3112).

[43] Rafiei M, Niknam T, Aghaei J, Shafie-khah M, Catalão JP. Probabilistic Load Forecasting using an Improved Wavelet Neural Network Trained by Generalized Extreme Learning Machine. IEEE Transactions on Smart Grid. 2018 Feb 21.

[44] Christiaanse WR. Short-term load forecasting using general exponential smoothing. IEEE Transactions on Power Apparatus and Systems. 1971 Mar(2):900-11.

[45] Ghelardoni L, Ghio A, Anguita D. Energy load forecasting using empirical mode decomposition and support vector regression. IEEE Trans. Smart Grid. 2013 Mar 1;4(1):549-56.

[46] Ceperic E, Ceperic V, Baric A. A strategy for short-term load forecasting by support vector regression machines. IEEE Transactions on Power Systems. 2013 Nov 1;28(4):4356-64.

[47] Sáez D, Ávila F, Olivares D, Cañizares C, Marín L. Fuzzy prediction interval models for forecasting renewable resources and loads in microgrids. IEEE Transactions on Smart Grid. 2015 Mar;6(2):548-56.

[48] Ertugrul ÖF. Forecasting electricity load by a novel recurrent extreme learning machines approach. International Journal of Electrical Power & Energy Systems. 2016 Jun 1;78:429-35.

[49] Hassan S, Khosravi A, Jaafar J, Khanesar MA. A systematic design of interval type-2 fuzzy logic system using extreme learning machine for electricity load demand forecasting. International Journal of Electrical Power & Energy Systems. 2016 Nov 1;82:1-0.

[50] Tripathi MM, Upadhyay KG, Singh SN. Short-term load forecasting using generalized regression and probabilistic neural networks in the electricity market. The Electricity Journal. 2008 Nov 1;21(9):24-34.

[51] Yadav HK, Pal Y, Tripathi MM. Photovoltaic power forecasting methods in smart power grid. InIndia Conference (INDICON), 2015 Annual IEEE 2015 Dec 17 (pp. 1-6). IEEE.