Auxiliary Network: Scalable and agile online learning for dynamic system with inconsistently available inputs

Rohit Agarwal[1[0000−0003−4846−7745]], Krishna Agarwal[1[0000−0001−6968−578X]], Alexander Horsch[1[0000−0001−7745−0139]], and Dilip K. Prasad[1[0000−0002−3693−6973]]

Bio-AI group, UiT The Arctic University of Norway, Hansine Hansens veg 18, 9019 Tromsø, Norway Corresponding: agarwal.102497@gmail.com

Abstract. Streaming classification methods assume the number of input features is fixed and always received. But in many real-world scenarios, some input features are reliable while others are unreliable or inconsistent. In this paper, we propose a novel deep learning-based model called Auxiliary Network (Aux-Net), which is scalable and agile. It employs a weighted ensemble of classifiers to give a final outcome. The Aux-Net model is based on the hedging algorithm and online gradient descent. It employs a model of varying depth in an online setting using single pass learning. Aux-Net is a foundational work towards scalable neural network for a dynamic complex environment requiring ad hoc or inconsistent inputs. The efficacy of Aux-Net is shown on a public dataset.

Keywords: Online Learning · Dynamic System · Inconsistent inputs

1 Introduction

Supporting varying number of input features can be a game changer in real life applications which deal with dynamic complex environments. Examples include a device in smart city environment, an autonomous vehicle, etc. To model such environment of inconsistent and scalable nature, we assume some reliable data channels and refer it as base input features, denoted by $\{x_1^B, \ldots, x_b^B, \ldots, x_B^B\}$. In addition, it may receive other information through auxiliary sensor arrays termed as auxiliary input features and denoted by $\{x_1^A, \ldots, x_a^A, \ldots, x_A^A\}$. Here, x denotes input features, B in superscript and subscript denotes base feature and the number of base features respectively. Similarly, A in superscript and subscript denotes auxiliary feature and the number of auxiliary features respectively. Due to the intermittent availability, only a subset of auxiliary features arrive along with the base features at any time instance t as shown in Figure 1. This imparts a lot of challenges as mentioned in Table 1. This problem can be approached via minimalist (only using base features), maximalist (making ensemble of model, one for each possible combinations of auxiliary features) and imputation based approaches (imputing the values wherever possible) but all the solutions are inefficient to address the problem. More information is given in Appendix 1.

An ideal solution would be an agile and scalable network architecture that adapts itself to the availability of auxiliary inputs without needing to maintain or train multiple networks or impute data. In this paper, we present a new paradigm of learning in the presence of inconsistently available auxiliary inputs, which we call auxiliary network (Aux-Net). The key-stone of Aux-net is the separation of learning corresponding to auxiliary and base inputs into separate modules parallel to each other (see Figure 2). The base features are processed as a chunk in base module while auxiliary module contains one independent layer per auxiliary input in parallel with other layers.

Table 1. Key differentiators of Aux-Net. ✓ *Fullsupport.*× No support. ∘ Partial support. Missing data: data for some time instances of an input feature is missing. Missing features: prior knowledge about their existence or distribution may be assumed (even if they arrive late). Obsolete features: features ceases to exists after some time. Sudden unknown features: no prior knowledge of their existence is available and they arrive late unannounced (true ad hoc). Unknown no. of features: no information about the number of inputs.

Characteristics	Aux-Net	ODL	Impute based
Online	✓	✓	✓
Missing data	✓	∘	✓
Missing features	✓	×	∘
Obsolete features	✓	×	∘
Sudden unknown features	✓	×	×
Unknown no. of features	✓	×	×

2 Related Works

Many methods based on Bayesian theory [14], k-nearest neighbour [1], support vector machines [15], decision tree [16], fuzzy logic [6, 2] are proposed for streaming classification task. A brief study of all these techniques can be found in [17, 18]. Furthermore, some incremental learning approaches are also proposed [7–11]. Other deep learning approaches for online learning include [19–22]. But these techniques assume the dimension of incoming data is fixed. Hou et al. [12, 13] proposed machine learning approaches for dynamic environments. However, they assumed the dimensionality of inputs is constant in batches and therefore batch wise learning can be used. Hou et al. [12] further assume that there are multiple sets of features, where one entire set is either available or unavailable in a given batch. [13] assumes an overlapping period between two batches when all the inputs from previous and next batches are available, which allows in supporting soft transition across batches. These methods are indeed more scalable than approaches that assume fixed input dimensionality. Nonetheless, they cannot handle as challenging situations as depicted in Figure 1 and discussed in Table 1 where no assumption is made on availability of auxiliary features in batches or sets. To the best of our knowledge, our work is a foundational work for problems of inconsistently available inputs. The only assumption is that at least one base feature is consistently available. Our work has a more general premise. We note that our framework is inspired from the concept of hedge algorithm [3] and online gradient descent (OGD) [4].

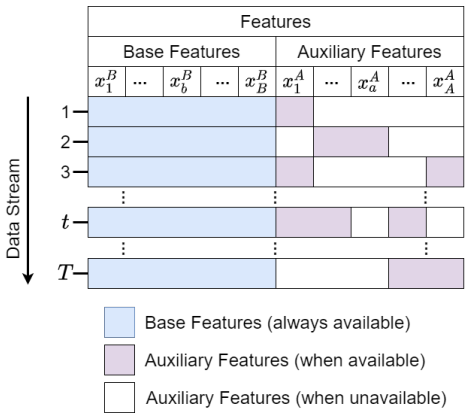

Fig. 1. Arrival of streaming data with all the base features and inconsistently available auxiliary features is demonstrated here.

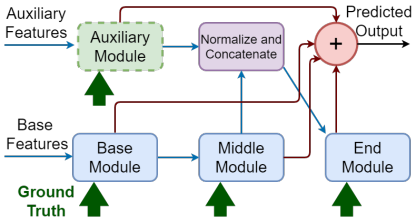

Fig. 2. Block diagram of Aux-Net. The green bold arrow represents the ground truth and the circle with a + sign calculates a final prediction from the weighted output of each classifier. All the modules are the combination of one or more hidden layers where the auxiliary module is scalable i.e. the number of layers keeps changing with time and scalability needs of the application.

3 Auxiliary Network (Aux-Net)

Problem Setting Let's denote streaming classification data by $D = \{(x_1, y_1), ..., (x_t, y_t), ..., (x_T, y_T)\}$ where $x_t = \{x_t^B, x_t^{A_t}\}$ is input at time t. The base features are denoted by $x_t^B = \{x_{1,t}^B, ..., x_{b,t}^B, ..., x_{B,t}^B\}$, where B in superscript and subscript denotes base features and total number of base features respectively. $x_{b,t}^B$ denotes b^{th} base feature at t. The auxiliary features is represented by $x_t^{A_t} = \{x_{j,t}^A\}_{\forall j \in A_t}$ where $A_t \subseteq \{1, ..., a, ...A\}$ is subset of auxiliary features received at t. The notation A has denotation similar to B. The input $x_t \in R^{d_t}$ where d_t is the dimension of x_t varying with t (Figure 1). The output $y_t \in R^C$ is the class label where C is the total number of classes. The Aux-Net learns a mapping $F : R^{d_t} \rightarrow R^C$. The prediction of the model is given by $\hat{y}_t = F(x_t)$. Model trains in an online setting where x_t arrives, it predicts $\hat{y}_t$, y_t is revealed and it is updated based on the loss.

Architecture Consider a DNN with S number of base layers, one middle layer, A number of auxiliary layers and E number of end layers. The base layers, middle layer, auxiliary layers and end layers constitute the base module, middle module, auxiliary module and end module respectively. The base, middle and end modules are stacked sequentially and auxiliary module is placed in parallel to the base and middle module with a connection to the end module as shown in the Figure 2. A softmax classifier is attached to each of the layer. The detailed architecture of the model is presented in Figure 3. The output of the Aux-Net model is given as the weighted combination of all the classifiers by the equation:

$$F(x) = \sum_{Z \in U} \sum_{z=1}^{Z} \alpha_z^Z f_z^Z \tag{1}$$

where $U = \{S, M, E, A\}$ denotes all the modules, and S, M, E, A in superscript and subscript denotes module name and the total number of layers in module

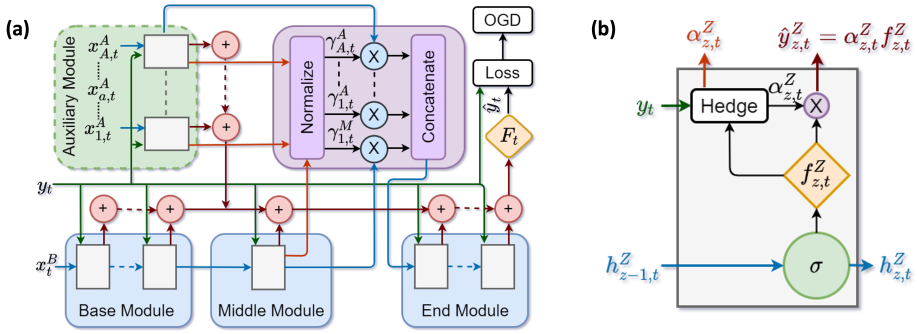

Fig. 3. (a) Detailed architecture of Aux-Net is presented here. The gray colored rectangular boxes represents a layer. (b) The functional diagram of a layer is shown here.

respectively. The notation f_z^Z and α_z^Z represents the output of the classifier associated with the layer z of module Z and weight of the classifier respectively.

Architecture of a layer is shown in Figure 3(b). Each layer is attached to a classifier f parameterized by θ that gives an output $f_z^Z = \text{softmax}(h_z^Z \theta_z^Z)$, where h_z^Z is the hidden feature of layer. Each layer is parameterized by W and c and generates its hidden feature $h_z^Z = \sigma(W_z^Z h_{z-1}^Z + c_z^Z)$, where σ is the activation function, and θ, W and c are learnt using OGD. A hedge block is used to compute α based on the loss incurred by the classifier.

Now, we describe the inputs to the different layers. The first base layer receives the complete x_t^B as the input i.e. $h_0^Z = x_t^B$. The subsequent base layers receive the hidden feature of its previous layer as its input. The input to middle layer is the hidden feature of the last base layer, i.e. $h_0^M = h_S^S$. The a^{th} auxiliary layer receives the a^{th} auxiliary feature, i.e., $h_a^A = x_a^A$. All the end layers, except the first end layer receive its previous layer features as the input. The first end layer is special since the input to it needs to support agility arising from only a subset of auxiliary features being available at time t. The input $h_{0,t}^E$ to the first end layer at time instance t is a vector derived by concatenating weighted hidden features h of the middle and the auxiliary layers corresponding to the currently available auxiliary inputs. It is given by $h_{0,t}^E = \left[\gamma_{1,t}^M h_{1,t}^M, \{\gamma_{j,t}^A h_{j,t}^A\}_{\forall j \epsilon A_t}\right]$, where γ is importance of the layers connected to first end layer, denoting the fraction of the connected layer's output passed as an input to the first end layer.

Parameters Learning The learning of the model occurs in an online setting through the use of a loss function defined as:

$$L(F(x), y) = \sum_{Z \epsilon U} \sum_{z=1}^{Z} \alpha_z^Z L(f_z^Z(x), y) \tag{2}$$

where $L(f_z^Z(x), y)$ is the loss of classifier associated with layer z of module Z. Based on the loss at each time step t, the values of $\gamma, \theta, W, c, \alpha$ are updated.

Updating γ : The highlight of Aux-Net is the update of γ which allows for soft handling of the asynchronous availability of auxiliary features. It depends

Algorithm 1: Aux-Net algorithm

Inputs: Base Module: S; Middle Module: M; Auxiliary Module: A; End Module: E; Learning rate: η; Smoothing Parameter: λ; Discounting Parameter: β ;

Initialize: A DNN with $L = S + M + A + E$ layers and attach classifiers to each layer as shown in Figure 3(b); $\alpha_z^Z = 1/L \quad \forall Z \in \{S, M, A, E\}, \quad z \in \{1, ..., Z\}$; K_1 using equation 7;

for $t = 1, ..., T$ **do**

Receive input feature x_t;
Create a list A_t of the auxiliary features received in x_t;
Create the model M_t based on A_t using equation 8;
Predict $\hat{y}_t$ on x_t using equation 1 based on M_t;
Receive output label y_t;
Calculate the loss of the model M_t based on y_t and $\hat{y}_t$ using equation 2;
Update parameters of M_t based on the loss incurred and get M_t^* using 9;
Update K_t based on M_t^* to get K_{t+1} using 10;

end

only on its classifiers weights and are calculated as follows:

$$\gamma_{p,t}^P = \frac{\alpha_{p,t}^P}{\alpha_{1,t}^M + \sum_{j \in A_t} \alpha_{j,t}^A} \quad \text{for} \quad \text{C1:}(P = M,\ p = 1) \text{ or } (\text{P = A},\ \text{p} \in \text{A}_\text{t}) \tag{3}$$

Updating θ : The parameter θ_z^Z is associated with only one classifier and does not depend on the other classifiers. Therefore, its update will only be with respect to the loss of its own classifier through OGD. After every time instance t, θ_z^Z of classifier z of the module Z is updated as:

$$\theta_{z,t+1}^Z = \theta_{z,t}^Z - \eta \alpha_{z,t}^Z \Delta_{\theta_{z,t}^Z, z}^Z \quad \text{for} \quad \text{C2:}(Z \in U',\ z \in \{1, ..., Z\}) \text{ or } (\text{Z = A},\ \text{z} \in \text{A}_\text{t}) \tag{4}$$

where, $\Delta_{\theta_{z,t}^Z, r}^R = \frac{\partial L(f_r^R(x_t), y_t)}{\partial \theta_{z,t}^Z}$, η is learning rate of parameters and $U' = \{S, M, E\}$.

Learning W and c : The weights (W) and bias (c) of a layer are learned by back propagation on the final loss similar to OGD. But, since each layer is associated with a classifier unlike the traditional DNN where only last layer gives a prediction, the gradient descent is different. Here, the parameters of a layer depends on loss of all its successive layers that directly or indirectly influence it. The following equation shows update rule for W and same is applicable for c.

$$\begin{aligned} W_{a,t+1}^A &= W_{a,t}^A - \eta \Big[\alpha_{a,t}^A \Delta_{W_{a,t}^A, a}^A + \sum_{e=1}^{E} \alpha_{e,t}^E \Delta_{W_{a,t}^A, e}^E \Big] \\ W_{z,t+1}^Z &= W_{z,t}^Z - \eta \Big[\sum_{j=z}^{Z} \alpha_{j,t}^Z \Delta_{W_{z,t}^Z, j}^Z + \sum_{Q=set} \sum_{q=1}^{Q} \alpha_{q,t}^Q \Delta_{W_{z,t}^Z, q}^Q \Big] \end{aligned} \tag{5}$$

where $set = \{M, E\}, \{E\}, \phi$ if $Z \in \{S\}, \{M\}, \{E\}$ respectively, and $z \in \{1, ..., Z\}$.

Learning α : The value of α is learned through hedge algorithm. Initially, α is uniformly distributed i.e., $\alpha_z^Z = 1/L$, where L is the total number of layers, $L = S + M + A + E$. The loss incurred by classifier z of module Z at time t is

$L(f_z^Z(x_t), y_t)$ and its weight is $\alpha_{z,t}^Z$. The weights of the classifier are updated as:

$$\alpha_{z,t+1}^Z = \alpha_{z,t}^Z \beta^{L(f_z^Z(x_t), y_t)} \qquad \text{for C2,} \tag{6}$$

where $\beta \in (0,1)$ is the discount rate parameter. To avoid situation, $\alpha_z^Z \to 0$ (since we don't want to neglect any layer), a smoothing parameter $\lambda \in (0,1)$ is introduced. It ensures minimum weight for each classifier by $\alpha_{z,t+1}^Z = \max(\alpha_{z,t+1}^Z, \lambda/\mathrm{L})$. The value of all α is then normalized such that $\sum_{Z=U'} \sum_{z=1}^{Z} \alpha_{z,t+1}^Z + \sum_{j \in A_t} \alpha_{j,t+1}^A = 1$.

Algorithm Aux-Net is a test-then-train approach and since auxiliary features are changing, the trained model learned at t can't be used as it is for training or testing at $t+1$. We define a knowledge base K (updated parameters represented by $'$) which is updated after every time t. K at any time instance t is given by

$$K_t = \{W_t', c_t', \theta_t', \alpha_t'\}, \text{ where } G_t = \{G_t^S, G_t^M, G_t^A, G_t^E\} \text{ if } G \in \{W', c', \theta', \alpha'\} \tag{7}$$

Before training or testing, model needs to incorporate the incoming dynamic auxiliary features. We define a model M_t (eq. 8), that handles the asynchronous availability of auxiliary features (A_t) by introducing the variable γ. The model M_t predicts an output $\hat{y_t}$, given x_t and updates its parameter giving M_t^* based on the loss incurred. Before moving to the next instance, we update the final parameters of K_t based on M_t^*, giving knowledge base K_{t+1} (see Algorithm 1).

Creating Model (M_t): Based on the auxiliary features A_t received at time step t and knowledge base K_t, the model M_t is created before prediction and training. The auxiliary layers corresponding to A_t are kept active and all the other auxiliary layers are freeze. Freezing of layers means all the parameters associated with this layer will not be trained (or removing the layer from the model). Since, some of the auxiliary layers are removed, the value α of the model changes and a parameter γ is introduced. The model M_t is given by:

$$M_t = M(W_t, c_t, \theta_t, \alpha_t, \gamma_t) \tag{8}$$

where $G_t = \{G_t'^S, G_t'^M, \{G_{j,t}'^A\}_{\forall j \in A_t}, G_t'^E\}$ if $G = \{W, c, \theta\}$, $\alpha_t = \{\alpha_{z,t}^Z\}_{\forall \text{ C2}}$ where $\alpha_{z,t}^Z = \alpha_{z,t}'^Z \Big/ \left[\sum_{Z=U'} \sum_{z=1}^{Z} \alpha_{z,t}'^Z + \sum_{j \in A_t} \alpha_{j,t}'^A \right]$, $\gamma_t = \{\gamma_{p,t}^P\}_{\forall \text{ C1}}$, $\gamma_{p,t}^P = \alpha_{p,t}'^P \Big/ \left[\alpha_{1,t}'^M + \sum_{j \in A_t} \alpha_{j,t}'^A \right]$.

Obtaining knowledge base K_{t+1} for next instance: M_t is updated based on loss incurred at time t. The updated model, represented by M_t^* is given by:

$$M_t^* = M(W_t^*, c_t^*, \theta_t^*, \alpha_t^*) \tag{9}$$

where $W_t^*, c_t^*, \theta_t^*, \alpha_t^*$ are the parameters obtained by updating the parameters $W_t, c_t, \theta_t, \alpha_t$ of the model M_t by using equation 2, 4, 5 and 6. After training the model at time step t, we create the knowledge base K_{t+1} before moving to the next iteration. All the parameters updated at time step t and the parameters of the freezed layers ($A - A_t$) are collected. Then, K_{t+1} is given by:

$$K_{t+1} = \{W_{t+1}', c_{t+1}', \theta_{t+1}', \alpha_{t+1}'\} \tag{10}$$

where $G_{t+1}' = \{G_t^*, \{G_{j,t}'^A\}_{\forall\, j \in A - A_t}\}$ if $G \in \{W, c, \theta\}$, $\alpha_{t+1}' = \{\alpha_{z,t+1}'^Z\}_{Z \in U,\, z=\{1,\ldots,Z\}}$ where $\alpha_{z,t+1}'^Z = \alpha_{z,t}''^Z \Big/ \left[\sum_{Z \in U} \sum_{z=1}^{Z} \alpha_{z,t}''^Z \right]$ and $\alpha_{t+1}'' = \{\alpha_t^*, \{\alpha_{j,t}'^A\}_{\forall\, j \in A - A_t}\}$.

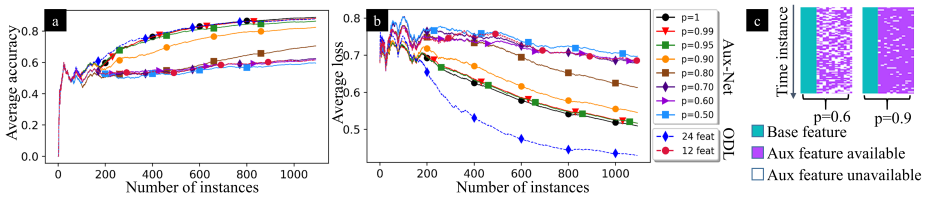

Fig. 4. Cumulative average accuracy (a) and loss (b) for different values of probability p of auxiliary inputs on Italy power demand dataset. ODL with 12 and 24 features is included for baseline. Snippet of data availability for $p = 0.6$ and $p = 0.9$ are shown in (c), analogous to Figure 1.

4 Experimental Results

We show robust, agile and scalable performance on Italy power demand dataset [5]. It has 1096 data instances with 24 features. In all the studies, we retain the original order of features. To the best of our knowledge, no method incorporates the intermittently available input data in an online setting. Thus, we compare Aux-Net with ODL [21] (in minimalist approach). We train both the models in a purely online setting where after each instance the model predicts and trains.

Architecture details The number of base layers (S) are 5, middle layer (M) is 1, and end layers (E) are 5 for Aux-Net. The number of auxiliary layers (A) are equal to number of auxiliary features. The number of layers for ODL is set as 11 ($S + E + M = 11$). For both Aux-Net and ODL, we used ReLU activation function, adam optimizer ($\eta = 0.01$), cross-entropy loss, smoothing rate ($\lambda = 0.2$), discount rate ($\beta = 0.99$) and number of nodes in each layer was set as 50.

Varying probability of the availability of auxiliary inputs in Aux-Net The first 12 features of Italy power demand dataset are considered as base features and remaining as auxiliary features. The availability of each auxiliary feature at a given time instance is modeled as a uniform distribution with probability p. The same value of p is used for all auxiliary features but the availability of each is computed independently. The results of Aux-Net with varying values of p, ODL with all 24 features, and 12 base features are presented in Table 2. We report the average of losses and accuracy observed across all time instances. The cumulative average loss and accuracy curves are shown in Figure 4. We study the performance of Aux-Net and compare with ODL with the following aims:

Sensitivity of Aux-Net to p and its performance: The average accuracy and loss for all the time instances in the dataset shows monotonic trend as a function of p, as noted in Table 2. This shows that Aux-Net is sensitive to the availability of the auxiliary inputs, as expected. Yet, the performance of Aux-Net degrades gracefully as p reduces. Moreover, Aux-Net still performs better compared to ODL with 12 features when $p < 1$ (as ODL can not work with inconsistent features). Further, the best case performance of Aux-Net when $p = 1$, is comparable to the scenario of ODL with 24 features.

This means that even though the knowledge base of Aux-Net supports for 2^{12} knowledge models, only the knowledge model with largest dimensionality is invoked and trained. In this case loss of Aux-Net is poorer than ODL, but the accuracy is better. In case of $p = 0.5$ which means no consistency in either availability or unavailability of the auxiliary inputs, the observed poorer performance of Aux-Net in comparison to ODL is only marginal, indicating robustness of Aux-Net to the extremely challenging scenario and its graceful degradation.

Table 2. Average accuracy and loss of Aux-Net (for different values of probability(p) of availability of auxiliary features) and ODL (with different number of input features(feat)) in Italy Power Demand dataset. *ODL* is shown in italics.

Model	Accuracy	Loss
ODL(24 feat)	*0.8783*	*0.4297*
Aux-Net($p = 1.00$)	0.8884	0.5093
Aux-Net($p = 0.99$)	0.8811	0.5165
Aux-Net($p = 0.95$)	0.8637	0.5168
Aux-Net($p = 0.90$)	0.8243	0.5456
Aux-Net($p = 0.80$)	0.7054	0.6130
Aux-Net($p = 0.70$)	0.6240	0.6788
Aux-Net($p = 0.60$)	0.6167	0.6831
ODL(12 feat)	*0.6139*	*0.6868*
Aux-Net($p = 0.50$)	0.5956	0.6975

Agile adaptation of Aux-Net: The demand on agility significantly enhances as p reduces. For example, for $p = 0.6$ in Figure 4(c), not only a different knowledge model needs to be invoked at every instance but also the same knowledge model may not be invoked in next many instances. The situation is easier when $p = 0.9$ even though there are many instances when a different knowledge model is invoked. Nonetheless, Aux-Net remains stable in either case and adapts to the agility needs in an efficient manner, indicated in accuracy and loss plots in Figure 4(a,b). Indeed, accuracy is better and loss decreases faster over time for $p = 0.9$. Nonetheless, when $p = 0.6$, the accuracy and loss curves closely follow ODL with 12 features, indicating that even though new knowledge models are being dynamically invoked every single instance, the performance of Aux-Net does not deteriorate in comparison to ODL and Aux-Net is indeed able to maintain agility over time, contributing to reduced loss and improved accuracy as time passes.

Decreased loss and improved accuracy over time: For situation of 12 auxiliary inputs, support for 2^{12} knowledge models, and invocation of each knowledge model multiple times is needed to study the convergence of knowledge base over time. Yet, the decreasing loss (Figure 4(b)) is a positive indicator of performance improvement over time and possible convergence.

Varying number of base features In this experiment, we fix p as 0.9, but vary the number of base features (B) from 1-23. The number of auxiliary features (A) are consequently (24-B). The first B features in the dataset are used as base features in Aux-Net and the only features in ODL. The average loss of Aux-Net and ODL are compared in Figure 5 as a function of B. We observe the following:

Extreme scalability: As expected, the performances of both Aux-Net and ODL deteriorate as B reduces. Nonetheless, the loss of Aux-Net is significantly smaller than ODL in the challenging scenarios when more than 4 inputs are inconsistently available. This clearly indicates that Aux-Net is able to leverage the auxiliary inputs for better learning. Especially, the extremely challenging

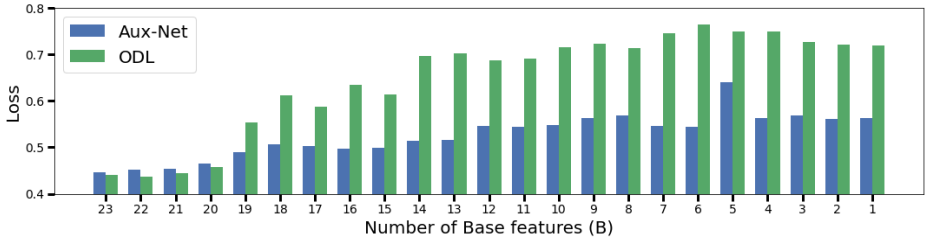

Fig. 5. Loss of Aux-Net as function of the number of base features B and ODL (trained using B number of features) in Italy power demand dataset. The probability of availability of the $(24 - B)$ auxiliary inputs is fixed at $p = 0.9$. Lower loss indicates better learning.

scenarios ($B = 1$ for example) demonstrate that Aux-Net is indeed able to step up to the need of supporting several knowledge models of varying inputs and dimensionalities and provide better performance than the minimalist approach.

Poorer performance than ODL when B $\in [20, 23]$**:** During initialization, Aux-Net assigns the same weights (α) to each classifier. However, the classifier corresponding to an auxiliary feature will be lossier as compared to the classifier of middle layer that uses base features. As time progresses, the value of α for each layer gets customized to suit its contribution towards accurate classification. Often, it means that α of auxiliary layer reduces in the first few time instances, indicating that Aux-Net has learnt that its inconsistent availability may cause increased loss if α corresponding to it is high.

Obsolete features In Figure 6, we consider an example in which all the auxiliary features (12 out of 24 features of the original dataset) are available initially, but become obsolete after the 100^{th} instances out of a total of 1096 instances (unavailable > 90% of the time). Therefore, the remaining 12 (base) features represent the data trend over the long term. In this condition, since we have the information about the obsolete features from their previously received instances, we can perform imputations. In the first 100 instances when all the features are available, ODL (mean imputation) and Aux-Net generate lower loss than ODL (12 features). However, after the 12 auxiliary features become obsolete, Aux-Net quickly adapts to the new normal and converges to similar performance as ODL (12 features). On the other hand, ODL (mean imputation) performs robustly for sometime due to imputation, but very slowly converges thereafter towards the new normal over the long term.

Sudden unknown features In Figure 7, we consider an example where only 6 (base) features are available. Suddenly, 18 new features with no prior knowledge appear from 201^{st} time instance. Aux-Net performs similar to ODL (6 features) till the 200^{th} instance. Thereafter, Aux-Net quickly adapts to the availability of new features and its loss starts decreasing at a rapid rate.

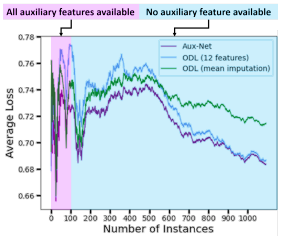

Fig. 6. All 12 auxiliary features become obsolete after 100th instance on Italy power demand dataset.

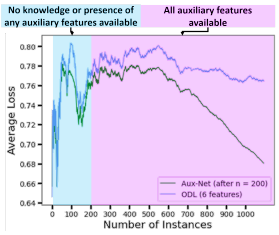

Fig. 7. 18 unknown auxiliary features starts appearing after 200th instance on Italy power demand dataset.

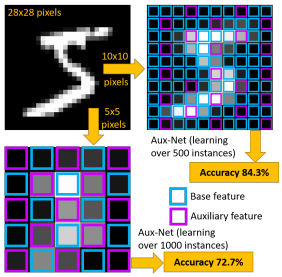

Fig. 8. Downsampled MNIST data with feature unavailability

MNIST dataset We applied AuxNet on MNIST [23] dataset for classification of 2 similar looking digits, namely '5' and '6' (see Figure 8) to demonstrate its performance on challening situations. In our version, we consider availability of a downsampled versions of the original 28×28 pixels. We consider two cases, 10×10 pixels and 5×5 pixels. In either case, the intensity at each pixel is one feature and a series of images is provided as data stream. For the case of 10×10 pixels, we consider 75:25 ratio of base and auxiliary features (blue and magenta colored boxes in Figure 8, respectively) and achieve classification accuracy of $> 84\%$. We further challenge Aux-Net with 12:13 ratio of base and auxiliary features for the case of 5×5 pixels, and achieve classification accuracy of $> 72\%$.

5 Conclusion

We have demonstrated scalability, agility, and stability of Aux-Net and its ability to deal with intermittently available inputs in an online setting. It supports scalability for the situations ranging from no auxiliary input being available to all auxiliary inputs being available. It incorporates knowledge models corresponding to all possible combinations of auxiliary inputs within a single knowledge base. The architectural support in Aux-Net for auxiliary inputs in the form of dedicated parallel layers is a critical feature for scalability. Agility in Aux-Net is characterized by its ability to dynamically invoke the relevant knowledge model without making the network unstable or unadaptive. A key factor that supports dynamic stability and agility is the importance parameter γ, which automatically adjusts the contributions of base inputs (through the middle layer) and the currently available auxiliary inputs so that neither the new auxiliary features introduce inordinate instability, nor are they suppressed. This, in our observation is not only the first such architecture, it is also a first demonstration of results on intermittently available input features. Having set a new paradigm, we hope that new datasets, frameworks, applications and more extensive studies are developed in future to exploit the possibility of learning in extremely dynamic and uncertain scenarios.

References

1. Aggarwal, Charu C., et al. "A framework for on-demand classification of evolving data streams." IEEE Transactions on Knowledge and Data Engineering 18.5 (2006): 577-589.
2. Iyer, Aparna Ramesh, Dilip K. Prasad, and Chai Hiok Quek. "PIE-RSPOP: A brain-inspired pseudo-incremental ensemble rough set pseudo-outer product fuzzy neural network." Expert Systems with Applications 95 (2018): 172-189.
3. Freund, Yoav, and Robert E. Schapire. "A decision-theoretic generalization of online learning and an application to boosting." Journal of computer and system sciences 55.1 (1997): 119-139.
4. Zinkevich, Martin. "Online convex programming and generalized infinitesimal gradient ascent." Proceedings of the 20th international conference on machine learning (icml-03). 2003.
5. Dau, Hoang Anh, et al. "The UCR time series archive." IEEE/CAA Journal of Automatica Sinica 6.6 (2019): 1293-1305.
6. Das, Ron Tor, Kai Keng Ang, and Chai Quek. "ieRSPOP: A novel incremental rough set-based pseudo outer-product with ensemble learning." Applied Soft Computing 46 (2016): 170-186.
7. Polikar, Robi, et al. "Learn++: An incremental learning algorithm for supervised neural networks." IEEE transactions on systems, man, and cybernetics, part C (applications and reviews) 31.4 (2001): 497-508.
8. Polikar, Robi, et al. "Learn++. MF: A random subspace approach for the missing feature problem." Pattern Recognition 43.11 (2010): 3817-3832.
9. Muhlbaier, Michael D., Apostolos Topalis, and Robi Polikar. "Learn $^{++}$. NC: Combining Ensemble of Classifiers With Dynamically Weighted Consult-and-Vote for Efficient Incremental Learning of New Classes." IEEE transactions on neural networks 20.1 (2008): 152-168.
10. Muhlbaier, Michael D., and Robi Polikar. "Multiple classifiers based incremental learning algorithm for learning in nonstationary environments." 2007 International conference on machine learning and cybernetics. Vol. 6. IEEE, 2007.
11. Ditzler, Gregory, Robi Polikar, and Nitesh Chawla. "An incremental learning algorithm for non-stationary environments and class imbalance." 2010 20th International Conference on Pattern Recognition. IEEE, 2010.
12. Hou, Chenping, and Zhi-Hua Zhou. "One-pass learning with incremental and decremental features." IEEE transactions on pattern analysis and machine intelligence 40.11 (2017): 2776-2792.
13. Hou, Bo-Jian, Lijun Zhang, and Zhi-Hua Zhou. "Learning with feature evolvable streams." Advances in Neural Information Processing Systems 30 (2017).
14. Seidl, Thomas, et al. "Indexing density models for incremental learning and anytime classification on data streams." Proceedings of the 12th international conference on extending database technology: advances in database technology. 2009.
15. Tsang, Ivor W., Andras Kocsor, and James T. Kwok. "Simpler core vector machines with enclosing balls." Proceedings of the 24th international conference on Machine learning. 2007.
16. Domingos, Pedro, and Geoff Hulten. "Mining high-speed data streams." Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. 2000.
17. Nguyen, Hai-Long, Yew-Kwong Woon, and Wee-Keong Ng. "A survey on data stream clustering and classification." Knowledge and information systems 45.3 (2015): 535-569.

18. Gama, Joao. "A survey on learning from data streams: current and future trends." Progress in Artificial Intelligence 1.1 (2012): 45-55.
19. Das, Monidipa, et al. "FERNN: A fast and evolving recurrent neural network model for streaming data classification." 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019.
20. Das, Monidipa, et al. "Muse-rnn: A multilayer self-evolving recurrent neural network for data stream classification." 2019 IEEE International Conference on Data Mining (ICDM). IEEE, 2019.
21. Sahoo, Doyen, et al. "Online deep learning: Learning deep neural networks on the fly." arXiv preprint arXiv:1711.03705 (2017).
22. Ashfahani, Andri, and Mahardhika Pratama. "Autonomous deep learning: Continual learning approach for dynamic environments." Proceedings of the 2019 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2019.
23. LeCun, Yann. "The MNIST database of handwritten digits." http://yann. lecun. com/exdb/mnist/ (1998).

Appendix 1

Minimalist: All uncertain inputs are dropped and a single knowledge model is trained using only base features. This model provides certain base accuracy, but does not utilize additional information from auxiliary inputs. The trade-off is the loss of opportunity for better performance.
Maximalist: An ensemble of 2^A networks can be formed to cater for all possible combinations of availability of auxiliary features. Therefore network with the smallest dimensionality caters to only the base features and network with the largest dimensionality caters to all the base and auxiliary features, where the number of inputs to a network is its dimensionality. However, learning the knowledge model in such ensemble of networks is cumbersome. Given A_t inputs features at time t, we have 2^{A_t} subsets, therefore the network corresponding to each subset needs to be trained. This results into long training durations. Another trade-off is that huge number of networks need to be maintained throughout.
Imputation: As shown in Table 1, it is possible to impute features whose prior information is known but it will introduce a lot of misinformation in cases where it is required to impute for many consecutive instances. Moreover, for the conditions of sudden unknown features, no prior information is available. So the approach can be to model noise (gaussian or otherwise) but we don't know the total number of features hence making it impossible to impute.