Online Deep Learning: Learning Deep Neural Networks on the Fly

Doyen Sahoo[1], Quang Pham[1], Jing Lu[2], Steven C. H. Hoi[1]
[1] School of Information Systems, Singapore Management Univerity,
[2] JD.com
{doyens,hqpham.2017}@smu.edu.sg, lvjing12@jd.com, chhoi@smu.edu.sg

Abstract

Deep Neural Networks (DNNs) are typically trained by backpropagation in a batch setting, requiring the entire training data to be made available prior to the learning task. This is not scalable for many real-world scenarios where new data arrives sequentially in a stream. We aim to address an open challenge of "Online Deep Learning" (ODL) for learning DNNs on the fly in an online setting. Unlike traditional online learning that often optimizes some convex objective function with respect to a shallow model (e.g., a linear/kernel-based hypothesis), ODL is more challenging as the optimization objective is non-convex, and regular DNN with standard backpropagation does not work well in practice for online settings. We present a new ODL framework that attempts to tackle the challenges by learning DNN models which dynamically adapt depth from a sequence of training data in an online learning setting. Specifically, we propose a novel Hedge Backpropagation (HBP) method for online updating the parameters of DNN effectively, and validate the efficacy on large data sets (both stationary and concept drifting scenarios).

1 Introduction

Despite the recent success of Deep Learning [LeCun *et al.*, 2015], it continues to face several (convergence) challenges, including (but not limited to) vanishing gradient, diminishing feature reuse [Srivastava *et al.*, 2015], saddle points (and local minima) [Dauphin *et al.*, 2014], immense number of parameters to be tuned, internal covariate shift [Ioffe and Szegedy, 2015], etc. There have been promising advances [Nair and Hinton, 2010; Ioffe and Szegedy, 2015; He *et al.*, 2016; Srivastava *et al.*, 2015], etc. to address many of these issues, however, most of them assume that the DNNs are trained in a batch learning setting which requires the entire training data set to be made available prior to the learning task. This is not possible for many real world tasks where data arrives sequentially in a stream, or may be too large to be stored in memory, or may exhibit concept drift [Gama *et al.*, 2014]. Thus, a more desired option is to learn the models in an online learning setting.

Unlike batch learning, online learning [Cesa-Bianchi and Lugosi, 2006] represents a class of learning algorithms that learn to optimize predictive models over a stream of data instances in a sequential manner. The nature of on-the-fly learning makes online learning highly scalable and memory efficient. However, most existing online learning algorithms are designed to learn shallow models (e.g., linear or kernel methods [Rosenblatt, 1958; Zinkevich, 2003; Crammer *et al.*, 2006; Kivinen *et al.*, 2004; Hoi *et al.*, 2013]) with online convex optimization, which cannot learn complex nonlinear functions in complicated application scenarios.

We attempt to bridge the gap between online and deep learning by addressing the open problem of "Online Deep Learning" (ODL) — how to learn DNNs in an online setting. A simple approach is to apply Backpropagation on a single instance in each online iteration - but this approach faces many limitations. A key challenge is to choose a proper model capacity (e.g. network depth) before starting to learn the model online. If the model is too complex (e.g., very deep), the learning process will converge too slowly (vanishing gradient, diminishing feature reuse, saddle points), thus losing the desired property of online learning. On the other extreme, if the model is too simple, the learning capacity will be too restricted, and without the power of depth, it would be difficult to learn complex patterns.

We aim to devise an online learning algorithm that starts with a shallow network enjoying fast convergence; then gradually switches to a deeper model (meanwhile sharing knowledge with the shallow ones) automatically when more data has been received to learn more complex hypotheses, combining the merits of both online learning and deep learning. To achieve this, we need to address the questions: *when* to change the network capacity? *how* to change the capacity? and how to do both *online*? We design an elegant solution to do this in a unified framework in a data-driven manner. We amend the existing DNN architecture by attaching every hidden layer representation to an output classifier. Then, instead of using a standard Backpropagation, we propose *Hedge Backpropagation*, which evaluates the performance of every output classifier at each online round using Hedge[Freund and Schapire, 1997], and appropriately extends Backpropagation to train DNNs online. This allows us to dynamically vary the DNN capacity, meanwhile enabling knowledge sharing between shallow and deep networks.

2 Related Work

Online Learning is a family of scalable algorithms that learn to update models from data streams sequentially [Cesa-Bianchi and Lugosi, 2006; Hoi *et al.*, 2014; 2018]. Popular algorithms include Perceptron [Rosenblatt, 1958], Online Gradient Descent [Zinkevich, 2003], Passive Aggressive (PA) [Crammer *et al.*, 2006], etc. These are primarily designed to learn linear models. Online Learning with kernels [Kivinen *et al.*, 2004] offered a solution for learning nonlinear models online, and were later extended to higher capacity models such as Online Multiple Kernel Learning [Hoi *et al.*, 2013; Sahoo *et al.*, 2014; 2016]. While these models learn nonlinearity, they are still shallow. Moreover, deciding the number and type of kernels is non-trivial; and these methods are not designed to *learn* a feature representation.

Online Learning can be directly applied to DNNs (”online backpropagation”) but they suffer from convergence issues(vanishing gradient, diminishing feature reuse, saddle points). Moreover, the optimal depth to be used for the network is usually unknown, and cannot be validated easily in the online setting. There have been attempts at making deep learning compatible with online learning [Zhou *et al.*, 2012; Lee *et al.*, 2016] and [Lee *et al.*, 2017]. However, they operate via a sliding window approach with a (mini)batch training stage, making them unsuitable for a streaming data setting.

Deep Learning Due to the difficulty in training deep networks, there has been a large body of emerging works adopting the principle of (what we term as)“shallow to deep”(used in our work). This approach exploits the intuition that shallow models converge faster than deeper models, and this idea has been executed in several ways. Some do this explicitly by Growing of Networks via the function preservation principle [Chen *et al.*, 2016; Wei *et al.*, 2016], where the (student) network of higher capacity is at least as good as the shallower (teacher) network. Other approaches perform this more implicitly by modifying the network architecture and objective functions to enable the network to allow the input to flow through the network, and slowly adapt to deep representation learning, e.g., Highway Nets[Srivastava *et al.*, 2015], Residual Nets[He *et al.*, 2016], Stochastic Depth Networks [Huang *et al.*, 2016] and Fractal Nets [Larsson *et al.*, 2017].

However, they are all designed to optimize the loss function based on the output of the deepest layer. Despite improved batch convergence, they cannot yield good online performance (particularly for early part of the stream), as many parameters need to be tuned. In online settings, such existing deep learning techniques could be trivially beaten by a very shallow network. Deeply Supervised Nets [Lee *et al.*, 2015] shares a similar architecture as ours - using companion objectives at intermediate layers with heuristically set weights. GoogLeNet [Szegedy *et al.*, 2015] also has intermediate classifiers, but the weights never change keeping the model capacity fixed, thus suitable only for batch settings. In contrast, our method dynamically adapts the model capacity. Since we aim to learn the depth, a related set of efforts is learning the architecture of neural networks [Zoph and Le, 2017; Alvarez and Salzmann, 2016] - which are all designed *only* for the batch setting.

3 Online Deep Learning

3.1 Problem Setting

Consider an online classification task. The goal of online deep learning is to learn a function $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^C$ based on a sequence of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, that arrive sequentially, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance representing the features and $y_t \in \{0,1\}^C$ is the class label assigned to $\mathbf{x}_t$ and C is the number of classes. The prediction is denoted by $\hat{y_t}$, and the performance of the learnt function is evaluated based on the cumulative prediction error: $\epsilon_T = \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}_{(\hat{y_t} \neq y_t)}$, where $\mathbb{I}$ is the indicator function. To minimize the classification error over the sequence of T instances, a loss function (e.g., squared loss, cross-entropy, etc.) is often chosen for minimization. In every online iteration, an instance $\mathbf{x}_t$ is observed, the model makes a prediction, the environment then reveals the true class label, and finally the learner makes an update to the model (e.g., using online gradient descent).

3.2 Online Backpropagation: Limitations

For typical online learning algorithms, the prediction function $\mathbf{F}$ is either a linear or kernel-based model. In the case of Deep Neural Networks (DNN), it is a set of stacked linear transformations, each followed by a nonlinear activation. Given an input $\mathbf{x} \in \mathbb{R}^d$, the prediction function of DNN with L hidden layers $(\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(L)})$ is recursively given by:

$$\begin{aligned} \mathbf{F}(\mathbf{x}) &= \text{softmax}(W^{(L+1)}\mathbf{h}^{(L)}) \quad \text{where} \\ \mathbf{h}^{(l)} &= \sigma(W^{(l)}\mathbf{h}^{(l-1)}) \quad \forall l = 1, \ldots, L; \quad \mathbf{h}^{(0)} = \mathbf{x} \end{aligned}$$

where σ is an activation function, e.g., sigmoid, tanh, ReLU, etc. This represents a feedforward step. The hidden layers $\mathbf{h}^{(l)}$ are feature representations learnt during training. To train a model with such a configuration, we use the cross-entropy loss function denoted by $\mathcal{L}(\mathbf{F}(\mathbf{x}), y)$. We aim to estimate the optimal model parameters W_i for $i = 1, \ldots (L+1)$ by applying Online Gradient Descent (OGD) on this loss function. Following the online learning setting, the update of the model in each iteration by OGD is given by:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \nabla_{W_t^{(l)}} \mathcal{L}(\mathbf{F}(\mathbf{x}_t), y_t) \quad \forall l = 1, \ldots, L+1$$

where η is the learning rate. Using backpropagation, the gradient of the loss with respect to $W^{(l)}$ for $l \leq L$ is computed.

Unfortunately, using such a model for online learning (i.e. Online Backpropagation) faces several issues with convergence. Most notably: (i) *Model Selection*: Depth of the network has to be fixed a priori, and cannot change. This is problematic as depth selection is a difficult task (especially for online settings). In particular for small number of instances, shallow networks would be preferred for fast convergence, and for large number of instances, deep networks could give the best overall performance; (ii) *Convergence Challenges*: These include vanishing gradient, saddle point problems and diminishing feature reuse (useful shallow features are lost in deep feedforward steps). These problems are more serious in the online setting (especially for the initial online performance), as we do not have the liberty to scan the data multiple times to overcome these issues (like we can in batch settings).

To address these issues, we design a training scheme for *Online Deep Learning* through a Hedging strategy: Hedge Backpropagation (HBP). Specifically, HBP uses an over-complete network, and automatically decides how and when to adapt the depth of the network in an online manner.

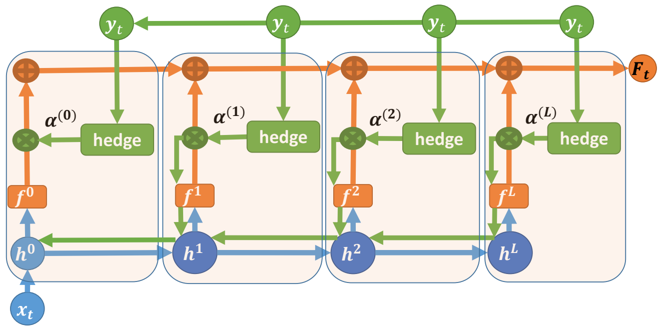


Figure 1: Online Deep Learning framework using Hedge Backpropagation (HBP). Blue lines represent feedforward flow for computing hidden layer features. Orange lines indicate softmax output followed by the hedging combination at prediction time. Green lines indicate the online updating flows with the hedge backpropagation approach.

3.3 Hedge Backpropagation (HBP)

Figure 1 illustrates the ODL framework using HBP.

Consider a DNN with L hidden layers (i.e., maximum capacity is L hidden layers). The prediction function for the proposed Hedged Deep Neural Network is given by:

$$\begin{aligned}\mathbf{F}(\mathbf{x}) &= \sum_{l=0}^{L} \alpha^{(l)}\mathbf{f}^{(l)} \quad \text{where} &(1)\\ \mathbf{f}^{(l)} &= \text{softmax}(\mathbf{h}^{(l)}\Theta^{(l)}),\ \forall l = 0,\dots,L\\ \mathbf{h}^{(l)} &= \sigma(W^{(l)}\mathbf{h}^{(l-1)}),\ \forall l = 1,\dots,L\\ \mathbf{h}^{(0)} &= \mathbf{x}\end{aligned}$$

Here we have designed a new architecture, and introduced two sets of new parameters $\Theta^{(l)}$ (parameters for $\mathbf{f}^{(l)}$) and α, that have to be learnt. Unlike the original network, in which the final prediction is given by a classifier using the feature representation $\mathbf{h}^{(L)}$, here the prediction is a weighted combination of classifiers learnt using the feature representations from $\mathbf{h}^{(0)},\dots,\mathbf{h}^{(L)}$. Each classifier $\mathbf{f}^{(l)}$ is parameterized by $\Theta^{(l)}$. Note that there are a total of $L+1$ classifiers. The final prediction of this model is a weighted combination of the predictions of all classifiers, where the weight of each classifier is denoted by $\alpha^{(l)} > 0$, and the loss suffered by the model is $\mathcal{L}(\mathbf{F}(\mathbf{x}),y) = \sum_{l=0}^{L}\alpha^{(l)}\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}),y)$. During the online learning procedure, we need to learn $\alpha^{(l)}$, $\Theta^{(l)}$ and $W^{(l)}$.

We propose to learn $\alpha^{(l)}$ using the Hedge Algorithm [Freund and Schapire, 1997]. At the first iteration, all weights α are uniformly distributed, i.e., $\alpha^{(l)} = \frac{1}{L+1}, l = 0,\dots,L$. At every iteration, the classifier $\mathbf{f}^{(l)}$ makes a prediction $\hat{y}_t^{(l)}$. When the ground truth is revealed, the classifier's weight is updated based on the loss suffered by the classifier as:

$$\alpha_{t+1}^{(l)} \leftarrow \alpha_t^{(l)}\beta^{\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}),y)}$$

where $\beta \in (0,1)$ is the discount rate parameter, and $\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}),y) \in (0,1)$ [Freund and Schapire, 1997]. Thus, a classifier's weight is discounted by a factor of $\beta^{\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}),y)}$ in every iteration. At the end of every round, the weights α are normalized such that $\sum_l \alpha_t^{(l)} = 1$.

Learning the parameters $\Theta^{(l)}$ for all the classifiers can be done via online gradient descent [Zinkevich, 2003], where the input to the l^{th} classifier is $\mathbf{h}^{(l)}$. This is similar to the update of the weights of the output layer in the original feedforward networks. This update is given by:

$$\begin{aligned}\Theta_{t+1}^{(l)} &\leftarrow \Theta_t^{(l)} - \eta\nabla_{\Theta_t^{(l)}}\mathcal{L}(\mathbf{F}(\mathbf{x}_t,y_t)) &(2)\\ &= \Theta_t^{(l)} - \eta\alpha^{(l)}\nabla_{\Theta_t^{(l)}}\mathcal{L}(\mathbf{f}^{(l)},y_t)\end{aligned}$$

Updating the feature representation parameters $W^{(l)}$ is more tricky. Unlike the original backpropagation scheme, where the error derivatives are backpropagated from the output layer, here, the error derivatives are backpropagated from *every* classifier $\mathbf{f}^{(l)}$. Thus, using the *dynamic objective* function $\mathcal{L}(\mathbf{F}(\mathbf{x}),y) = \sum_{l=0}^{L}\alpha^{(l)}\mathcal{L}(\mathbf{f}^{(l)}(\mathbf{x}),y)$ and applying OGD rule, the update rule for $W^{(l)}$ is given by:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta\sum_{j=l}^{L}\alpha^{(j)}\nabla_{W^{(l)}}\mathcal{L}(\mathbf{f}^{(j)},y_t) \qquad (3)$$

where $\nabla_{W^{(l)}}\mathcal{L}(\mathbf{f}^{(j)},y_t)$ is computed via backpropagation from error derivatives of $\mathbf{f}^{(j)}$. Note that the summation (in the gradient term) starts at $j = l$ because the shallower classifiers do not depend on $W^{(l)}$ for making predictions. In effect, we are computing the gradient of the final prediction with respect to the backpropagated derivatives of a predictor at every depth weighted by $\alpha^{(l)}$ (which is an indicator of the performance of the classifier). Hedge enjoys a regret of $R_T \le \sqrt{T\ln N}$, where N is the number of experts [Freund and Schapire, 1999], which in our case is the network depth. This gives an effective model selection approach to adapt to the optimal network depth automatically online.

Based on the intuition that shallower models tend to converge faster than deeper models [Chen *et al.*, 2016; Larsson *et al.*, 2017; Gulcehre *et al.*, 2016], using a hedging strategy would lower α weights of deeper classifiers to a very small value (due to poor initial performance as compared to shallower classifiers), which would affect the update in Eq. (3), and result in deeper classifiers having slow learning. To alleviate this, we introduce a *smoothing* parameter $s \in (0,1)$ which sets a minimum weight for each classifier. After the weight update of the classifiers in each iteration, the weights are set as: $\alpha^{(l)} \leftarrow \max\left(\alpha^{(l)},\frac{s}{L}\right)$ This helps us achieve a tradeoff between exploration and exploitation. s encourages all classifiers at every depth to affect the backprop update (exploring high capacity deep classifiers, and enabling deep classifiers to perform as good as shallow ones), while hedging the model exploits the best performing classifier. Similar strategies have been used in Multi-arm bandit setting, and online learning with expert advice to trade off exploration and exploitation [Auer *et al.*, 2002; Hoi *et al.*, 2013]. Algorithm 1 outlines ODL using HBP.

Algorithm 1 Online Deep Learning (ODL) using HBP

```
Inputs: Hedge: $\beta \in (0, 1)$; Learning Rate: $\eta$; Smoothing: $s$
**Initialize**: $\mathbf{F}(\mathbf{x}) =$ DNN with $L$ hidden layers and $L + 1$ classifiers $\mathbf{f}^{(l)}, \forall l = 0, \ldots, L$; $\alpha^{(l)} = \frac{1}{L+1}, \forall l = 0, \ldots, L$
  **for** t = 1,...,T **do**
    Receive instance: $\mathbf{x}_t$
    Predict $\hat{y}_t = \mathbf{F}_t(\mathbf{x}_t) = \sum_{l=0}^{L} \alpha_t^{(l)} \mathbf{f}_t^{(l)}$ as per Eq. (1)
    Reveal $y_t$ ; Set $\mathcal{L}_t^{(l)} = \mathcal{L}(\mathbf{f}_t^{(l)}(\mathbf{x}_t), y_t), \forall l, \ldots, L$;
    Update $\Theta_{t+1}^{(l)}$ & $W_{t+1}^{(l)} \forall l = 0, ..., L$ as per Eq. (2) & (3);
    Update $\alpha_{t+1}^{(l)} = \alpha_t^{(l)} \beta^{\mathcal{L}_t^{(l)}}, \forall l = 0, \ldots, L$;
    Smoothing $\alpha_{t+1}^{(l)} = \max(\alpha_{t+1}^{(l)}, \frac{s}{L}), \forall l = 0, \ldots, L$ ;
    Normalize $\alpha_{t+1}^{(l)} = \frac{\alpha_{t+1}^{(l)}}{Z_t}$ where $Z_t = \sum_{l=0}^{L} \alpha_{t+1}^{(l)}$
  **end for**
```

3.4 Discussions

There are several ideas and perspectives our proposed approach to Online Deep learning can be related to. We discuss some of these ideas below: (i) *Dynamic Objective*: Having a dynamically adaptive objective function mitigates the impact of vanishing gradient and helps escape saddle points and local minima (by changing the objective function without loss of performance). The multi-depth architecture also allows direct usage of intermediate features for prediction, mitigating diminishing feature reuse. In addition to being a solution for addressing Online Deep Learning, this idea can be applied to many other problem settings, where it may be difficult to design an appropriate objective function. In such settings, HBP can be applied to *learn the objective function* in a data-driven manner. A related concept is ResNet with Stochastic Depth [Huang *et al.*, 2016], where layers are arbitrarily dropped, and the effective network of the depth changes during the training procedure; (ii) *Online Learning with Expert Advice*,[Cesa-Bianchi and Lugosi, 2006]: In the proposed Online Deep Learning solution, the experts are DNNs of varying depth, and the HBP adaptation chooses the appropriate depth expert, making the DNN robust to depth of the network; (iii) *Student-teacher learning*[Chen *et al.*, 2016; Wei *et al.*, 2016]: Deep Networks would typically struggle to converge quickly, but as they are supported by the shallower networks when using HBP, they inherit a good initialization of shallow teacher networks; (iv) *Ensemble*: Multiple DNNs of varying depths compete (by Hedging) and collaborate (parameter sharing) for improved performance; (v) *Concept drifting*[Gama *et al.*, 2014]: HBP enables quick adaptation to new patterns due to hedging, and thus enables usage of DNNs for scenarios with concept drifts; and (vi) *Convolutional Networks*: While HBP could be trivially adapted to CNNs, computer vision tasks typically have many classes with few instances per class, which makes it hard to obtain robust results in just one-pass through the data (online setting — where train and test data is the same). These are inherently batch learning tasks. Our focus is on pure online settings where a large number of instances arrive in a stream and exhibit complex nonlinear patterns.

4 Experiments

4.1 Datasets

We consider several large scale datasets. Higgs and Susy are Physics datasets from UCI repository. For Higgs, we sampled 5 million instances. We used 5 million instances from Infinite MNIST [Loosli *et al.*, 2007]. We also evaluated on 3 synthetic datasets. Syn8 is generated from a randomly initialized DNN comprising 8-hidden layers (of width 100 each). The others are concept drift datasets CD1 and CD2. In CD1, 2 concepts (C1 and C2), appear in the form C1-C2-C1, with each segment comprising a third of the data stream. Both C1 and C2 were generated from a 8-hidden layer network. CD2 has 3 concepts with C1-C2-C3, where C1 and C3 are generated from a 8-hidden layer network, and C2 from a shallower 6-hidden layer network. Other details are in Table 1.

Data	#Features	#Instances	Type
Higgs	28	5m	Stationary
Susy	18	5m	Stationary
i-mnist	784	5m	Stationary
Syn8	50	5m	Stationary
CD1	50	4.5m	Concept Drift
CD2	50	4.5m	Concept Drift

Table 1: Datasets

4.2 Online BP Limitations: Depth Selection

We compare the online performance of DNNs of varying depth. Specifically, we compare their error rate in different windows (or stages) of the learning process. See Table 3. In the first 0.5% of data, the shallowest network obtains the best performance indicating faster convergence (suggesting we should use the shallow network for the task). In the segment [10-15]%, a 4-layer DNN seems to have the best performance in most cases. And in the segment from [60-80]% of data, an 8-layer network gives a better performance. This suggests that deeper networks took a longer time to converge, but at a later stage gave a better performance. Looking at the final error, it does not give us conclusive evidence of what depth of network would be the most suitable. Furthermore, if the datastream had more instances, an even deeper network may have given an overall better performance. This demonstrates the difficulty in model selection for learning DNNs online, where typical validation techniques are ineffective. Ideally we want to exploit fast convergence of shallow DNNs in the beginning and the power of deeper representations later.

4.3 Baselines

We aim to learn a 20 layer DNN in the online setting, with 100 units in each hidden layer. As baselines, we learn the 20 layer network online using OGD (Online Backpropagation), OGD Momentum, OGD Nesterov, and Highway Networks. We also compared with Online BP on DNNs with fewer layers (2,3,4,8,16) to get an idea of comparison with the oracle depth — as the best depth choice is task dependent and can be known only in hindsight. Configuration across all methods: ReLU activation, fixed learning rate of 0.01 (finetuned on the baselines). For momentum, a fixed learning rate of

0.001 was used, and momentum parameters were finetuned to give the best performance on the baselines. For HBP, we set $\beta = 0.99$ and the smoothing parameter $s = 0.2$. Implementation was in Keras [Chollet, 2015] [1]. We also compared with representative state of the art linear online algorithms (OGD, Adaptive Regularization of Weights (AROW), Soft-Confidence Weighted Learning (SCW) [Hoi *et al.*, 2014]) and kernel online algorithms (Fourier OGD (FOGD) and Nyström OGD (NOGD)[Lu *et al.*, 2016]).

4.4 Evaluation of ODL Algorithms

The final cumulative error obtained by all the baselines and the proposed HBP can be seen in Table 4. First, traditional online learning algorithms (linear and kernel) have relatively poor performance on complex datasets. Next, in learning with a 20-layer network, the convergence is slow, resulting in poor overall performance. While second order methods utilizing momentum and highway networks are able to offer some advantage over simple Online Gradient Descent, they can be easily beaten by a relatively shallower networks in the online setting. We observed before that relatively shallower networks could give a competitive performance in the online setting, but lacked the ability to exploit the power of depth at a later stage. In contrast, HBP enjoyed the best of both worlds, by allowing for faster convergence initially, and making use of the power of depth at a later stage. This way HBP was able to do automatic model selection online, enjoying merits of both shallow and deep networks, and this resulted in HBP outperforming all the DNNs of different depths, in terms of online performance. It should be noted that the optimal depth for DNN is not known before the learning process, and even then HBP outperforms all DNNs at any depth. Figure 3 shows convergence behavior of all the algorithms on the stationary as well as concept drift datasets. In the stationary datasets, HBP shows consistent outperformance over all the baselines. The only exception is in the very initial stages of the online learning phase, where shallower baselines are able to outperform HBP. This is not surprising, as HBP has many more parameters to learn. However, HBP is able to quickly outperform the shallow networks. The performance of HBP in concept drifting scenarios demonstrates its ability to adapt to change quickly, enabling usage of DNNs in the concept drifting scenarios. Looking at the performance of simple 20-layer (and 16-layer) networks on concept drifting data, we can see difficulty in utilizing deep representation for such scenarios.

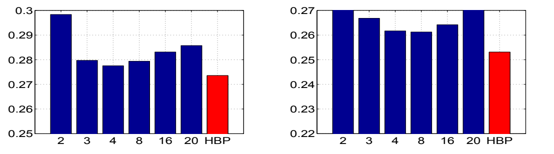
4.5 Adapting the Effective Depth of the DNN

We observe the evolution of weight distribution learnt by HBP over time in Figure 4. Initially (first 0.5%), the maximum weight has gone to the shallowest classifier (with just one hidden layer). In the second phase (10-15%), slightly deeper classifiers (classifiers with 4-5 layers) have picked up some weight, and in the third segment (60-80%), even deeper classifiers have gotten more weight (classifiers with 5-7 layers). The shallow and very deep classifiers receive little weight in the last segment showing HBPs ability to perform model selection.

[1]Source code available at https://github.com/LIBOL/ODL

4.6 Performance in Different Learning Stages

We compare the performance of HBP with DNNs of different depth in different stages of learning. Figure 2 shows that HBP matches (and even beats) the performance of the best depth network in both the beginning and at a later stage of the training phase. This shows its ability to exploit faster convergence of shallow networks in the beginning, and power of deep representation later. Not only is it able to do automatic model selection, but also it is able to offer a good initialization for the deeper representation, so that the depth of the network can be exploited sooner, thus beating a DNN of every depth.



(a) Error in 10-15% of data (b) Error in 60-80% of data

Figure 2: Error Rate in different segments of the Data. Red represents HBP using a 20-layer network. Blue are OGD using DNN with layers = 2,3,4,8, 16 and 20.

4.7 Robustness to Depth of Base-Network

We evaluate HBP with varying depth of the base network. We consider 12, 16, 20, and a 30-layer DNNs trained using HBP and Online BP on Higgs. See Table 2 for the results, where the performance variation with depth does not significantly alter HBP's performance, while for simple Online BP, increase in depth significantly hurts the learning process. This shows that, despite an arbitrary depth base-DNN, HBP mitigates several shortcomings of traditional DNN's, and consistently gives a good performance.

Depth	**12**	**16**	**20**	**30**
OnlineBP	26.96$\pm$0.07	27.31$\pm$0.13	29.27$\pm$0.65	47.67$\pm$0.01
HBP	26.21$\pm$0.03	26.18 $\pm$0.04	26.18$\pm$0.03	26.23$\pm$0.04

Table 2: Robustness of HBP to depth of the base network

5 Conclusion

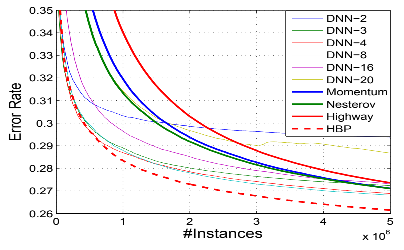
This paper addressed the critical drawbacks of existing DNNs when being used to learn from streaming data in an online setting. These issues arose from difficulty in model selection (appropriate depth), and convergence difficulties (vanishing gradient, saddle points & diminishing feature reuse). We used the "shallow to deep" principle, and devised the Hedge Backpropagation method, which enabled on-the-fly training of Deep Neural Networks in an online setting. HBP used a hedging strategy to make predictions with multiple outputs from different hidden layers of the network, and the backpropagation algorithm was modified to allow for knowledge sharing among the deeper and shallower networks. This approach automatically identified how and when to modify the effective network capacity in a data-drive manner, based on the observed data complexity. We validated the proposed method through extensive experiments on large datasets.

	Final Cumulative Error			**Segment [0-0.5]% Error**			**Segment [10-15]% Error**			**Segment [60-80]% Error**		
L	**Higgs**	**Susy**	**Syn8**	**Higgs**	**Susy**	**Syn8**	**Higgs**	**Susy**	**Syn8**	**Higgs**	**Susy**	**Syn8**
3	27.24	20.16	39.36	**35.84**	**21.52**	**42.69**	27.97	**20.29**	40.02	26.68	20.04	39.01
4	26.88	**20.14**	**39.20**	37.21	21.97	43.39	**27.75**	**20.30**	**39.89**	26.17	20.04	**38.76**
8	**26.82**	20.16	39.36	38.08	22.18	45.22	27.94	20.36	40.18	**26.13**	**19.97**	38.88
16	27.31	20.37	40.25	45.50	23.12	47.21	28.31	20.50	41.21	26.42	20.27	39.23

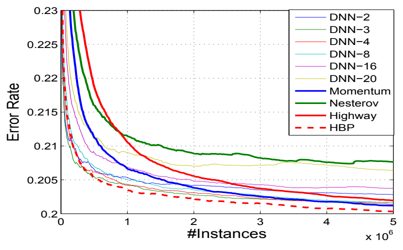
Table 3: Online error rate (%) of DNNs of varying Depth in different stages of learning. L is the number of layers in the DNN.

Model	**Method**	**L**	**Higgs**	**Susy**	**i-mnist**	**Syn8**	**CD1**	**CD2**
Linear	**OGD**	1	$36.20_{\pm 0.200}$	$21.70_{\pm 0.200}$	$12.30_{\pm 0.200}$	$40.70_{\pm 0.200}$	$43.60_{\pm 0.200}$	$42.70_{\pm 0.100}$
	AROW	1	$36.30_{\pm 0.100}$	$21.60_{\pm 0.200}$	$12.40_{\pm 0.200}$	$40.50_{\pm 0.100}$	$43.40_{\pm 0.100}$	$42.50_{\pm 0.200}$
	SCW	1	$35.30_{\pm 0.100}$	$21.50_{\pm 0.200}$	$12.30_{\pm 0.100}$	$40.50_{\pm 0.100}$	$43.40_{\pm 0.100}$	$42.50_{\pm 0.100}$
Kernel	**FOGD**	2	$29.74_{\pm 0.003}$	$20.21_{\pm 0.002}$	$4.96_{\pm 0.004}$	$39.62_{\pm 0.003}$	$43.29_{\pm 0.001}$	$41.91_{\pm 0.004}$
	NOGD	2	$34.87_{\pm 0.003}$	$20.45_{\pm 0.001}$	$10.45_{\pm 0.001}$	$41.47_{\pm 0.002}$	$44.55_{\pm 0.004}$	$43.57_{\pm 0.003}$
DNNs	**OGD (Online BP)**	2	$29.38_{\pm 0.039}$	$20.29_{\pm 0.004}$	$1.98_{\pm 0.018}$	$39.73_{\pm 0.030}$	$41.51_{\pm 0.047}$	$37.23_{\pm 0.021}$
	OGD (Online BP)	3	$27.25_{\pm 0.017}$	$20.15_{\pm 0.010}$	$1.93_{\pm 0.017}$	$39.30_{\pm 0.019}$	$41.12_{\pm 0.022}$	$37.09_{\pm 0.037}$
	OGD (Online BP)	4	$26.88_{\pm 0.044}$	$20.14_{\pm 0.016}$	$1.93_{\pm 0.047}$	$39.19_{\pm 0.043}$	$41.13_{\pm 0.036}$	$37.05_{\pm 0.033}$
	OGD (Online BP)	8	$26.79_{\pm 0.046}$	$20.17_{\pm 0.004}$	$3.19_{\pm 1.997}$	$39.41_{\pm 0.018}$	$41.42_{\pm 0.025}$	$37.33_{\pm 0.074}$
	OGD (Online BP)	16	$27.43_{\pm 0.169}$	$20.39_{\pm 0.029}$	$2.22_{\pm 0.064}$	$40.90_{\pm 1.499}$	$43.14_{\pm 1.353}$	$38.38_{\pm 0.223}$
	OGD (Online BP)	20	$29.27_{\pm 0.655}$	$20.61_{\pm 0.063}$	$2.62_{\pm 0.074}$	$46.37_{\pm 2.529}$	$48.37_{\pm 1.823}$	$48.69_{\pm 0.000}$
	OGD+Momentum	20	$27.13_{\pm 0.086}$	$20.09_{\pm 0.008}$	$2.80_{\pm 0.139}$	$39.69_{\pm 0.186}$	$42.83_{\pm 0.719}$	$38.39_{\pm 1.002}$
	OGD+Nesterov	20	$26.94_{\pm 0.058}$	$20.08_{\pm 0.018}$	$2.75_{\pm 0.147}$	$39.94_{\pm 0.185}$	$43.23_{\pm 0.648}$	$39.43_{\pm 4.297}$
	Highway	20	$27.94_{\pm 0.544}$	$20.76_{\pm 0.520}$	$2.79_{\pm 0.263}$	$46.92_{\pm 0.877}$	$49.28_{\pm 0.000}$	$46.03_{\pm 2.926}$
	HBP (proposed)	20	$\mathbf{26.18_{\pm 0.030}}$	$\mathbf{20.03_{\pm 0.005}}$	$\mathbf{1.56_{\pm 0.020}}$	$\mathbf{38.96_{\pm 0.047}}$	$\mathbf{40.82_{\pm 0.033}}$	$\mathbf{36.72_{\pm 0.047}}$

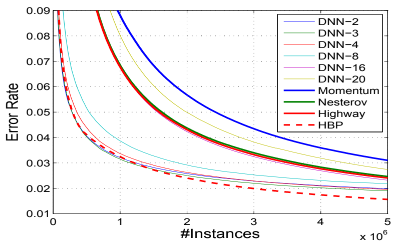
Table 4: Final Online Cumulative Error Rate (%) obtained by the algorithms (± indicates standard deviation). Best performance is in bold.



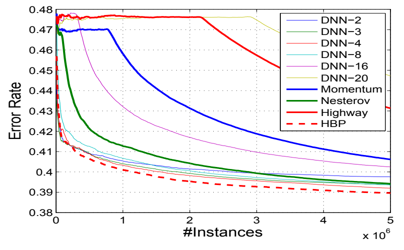
(a) HIGGS



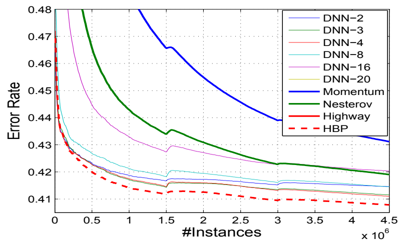
(b) SUSY



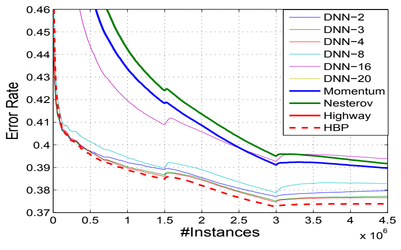
(c) Inf-MNIST (i-mnist)



(d) SYN8

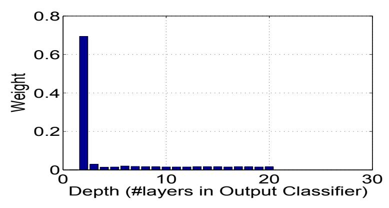


(e) Concept Drift 1 (CD1)

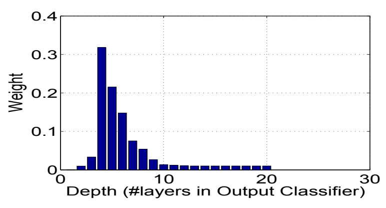


(f) Concept Drift 2 (CD2)

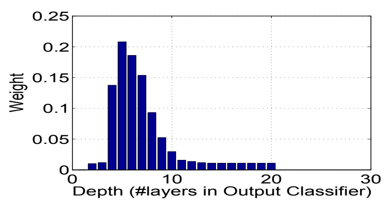
Figure 3: Convergence behavior of DNNs in Online Setting on stationary and concept drifting data.



(a) First 0.5% of Data



(b) 10-15% of Data



(c) 60-80% of Data

Figure 4: Evolution of weight distribution of the classifiers over time using HBP on HIGGS dataset.

Acknowledgements

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

[Alvarez and Salzmann, 2016] JM Alvarez and M Salzmann. Learning the number of neurons in deep networks. In *NIPS*, 2016.

[Auer *et al.*, 2002] N Auer, Pand Cesa-Bianchi, Y Freund, and RE Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 2002.

[Cesa-Bianchi and Lugosi, 2006] N Cesa-Bianchi and G Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[Chen *et al.*, 2016] T Chen, I Goodfellow, and J Shlens. Net2net: Accelerating learning via knowledge transfer. *ICLR*, 2016.

[Chollet, 2015] François Chollet. Keras, 2015.

[Crammer *et al.*, 2006] K Crammer, O Dekel, J Keshet, S Shalev-Shwartz, and Y Singer. Online passive-aggressive algorithms. *JMLR*, 2006.

[Dauphin *et al.*, 2014] Y Dauphin, R Pascanu, C Gulcehre, K Cho, S Ganguli, and Y Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2014.

[Freund and Schapire, 1997] Y Freund and RE Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS*, 1997.

[Freund and Schapire, 1999] Y Freund and RE Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 1999.

[Gama *et al.*, 2014] J Gama, I Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *CSUR*, 2014.

[Gulcehre *et al.*, 2016] C Gulcehre, M Moczulski, F Visin, and Y Bengio. Mollifying networks. *preprint arXiv:1608.04980*, 2016.

[He *et al.*, 2016] K He, X Zhang, S Ren, and J Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[Hoi *et al.*, 2013] S.C.H. Hoi, R Jin, P Zhao, and T Yang. Online multiple kernel classification. *Machine Learning*, 2013.

[Hoi *et al.*, 2014] S.C.H. Hoi, J Wang, and P Zhao. Libol: A library for online learning algorithms. *Journal of Machine Learning Research*, 2014.

[Hoi *et al.*, 2018] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao. Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871*, 2018.

[Huang *et al.*, 2016] G Huang, Y Sun, Z Liu, D Sedra, and K Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[Ioffe and Szegedy, 2015] S Ioffe and C Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[Kivinen *et al.*, 2004] J. Kivinen, A.J. Smola, and R.C. Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 2004.

[Larsson *et al.*, 2017] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *ICLR*, 2017.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[Lee *et al.*, 2015] C Lee, S Xie, P Gallagher, Z Zhang, and Z Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[Lee *et al.*, 2016] S Lee, C Lee, D Kwak, J Kim, J Kim, and B Zhang. Dual-memory deep learning architectures for lifelong learning of everyday human behaviors. In *IJCAI*, 2016.

[Lee *et al.*, 2017] J Lee, J Yun, S Hwang, and E Yang. Lifelong learning with dynamically expandable networks. *arXiv:1708.01547*, 2017.

[Loosli *et al.*, 2007] G Loosli, S Canu, and L Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, pages 301–320, 2007.

[Lu *et al.*, 2016] J Lu, S.C.H. Hoi, J Wang, P Zhao, and Z Liu. Large scale online kernel learning. *Journal of Machine Learning Research*, 2016.

[Nair and Hinton, 2010] V Nair and GE Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[Rosenblatt, 1958] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[Sahoo *et al.*, 2014] D Sahoo, S. C. H. Hoi, and B Li. Online multiple kernel regression. In *ACM SIGKDD*, 2014.

[Sahoo *et al.*, 2016] D Sahoo, S. C. H. Hoi, and P Zhao. Cost sensitive online multiple kernel classification. In *ACML*, 2016.

[Srivastava *et al.*, 2015] RK Srivastava, K Greff, and J Schmidhuber. Training very deep networks. In *NIPS*, 2015.

[Szegedy *et al.*, 2015] C Szegedy, W Liu, Y Jia, P Sermanet, S Reed, D Anguelov, D Erhan, V Vanhoucke, and A Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[Wei *et al.*, 2016] Tao Wei, Changhu Wang, Rong Rui, and Chang Wen Chen. Network morphism. *ICML*, 2016.

[Zhou *et al.*, 2012] G Zhou, K Sohn, and H Lee. Online incremental feature learning with denoising autoencoders. *AISTATS*, 2012.

[Zinkevich, 2003] M Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.

[Zoph and Le, 2017] B Zoph and QV Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.