MODL: Multilearner Online Deep Learning

Antonios Valkanas[†]
McGill University, Mila[⋆], ILLS[⋆]

Boris N. Oreshkin[‡]
Amazon Science

Mark Coates
McGill University, Mila[⋆], ILLS[⋆]

Abstract

Online deep learning tackles the challenge of learning from data streams by balancing two competing goals: fast learning and deep learning. However, existing research primarily emphasizes deep learning solutions, which are more adept at handling the "deep" aspect than the "fast" aspect of online learning. In this work, we introduce an alternative paradigm through a hybrid multilearner approach. We begin by developing a fast online logistic regression learner, which operates without relying on backpropagation. It leverages closed-form recursive updates of model parameters, efficiently addressing the fast learning component of the online learning challenge. This approach is further integrated with a cascaded multilearner design, where shallow and deep learners are co-trained in a cooperative, synergistic manner to solve the online learning problem. We demonstrate that this approach achieves state-of-the-art performance on standard online learning datasets. We make our code available: https://github.com/AntonValk/MODL

1 INTRODUCTION

Off-line machine learning algorithms are trained on bulk datasets, making multiple passes over them to gradually tune model parameters. In numerous scenarios, it is advantageous to learn from data streams by processing each instance sequentially, giving rise to a method known as *Online Learning* (Bottou, 1998). Compared to batch training, online learning enables scalable and memory-efficient training, even with unbounded dataset sizes. Research in online learning spans several decades and encompasses supervised (Zinkevich, 2003), semi-supervised (Belkin et al., 2006), and unsupervised learning (Guha et al., 2000) archetypes. Early methods concentrated on shallow learners, which offer fast learning but limited expressive capacity (Hoi et al., 2013). In contrast, deep neural networks provide extensive expressive power, though their learning process is typically much slower. Sahoo et al. (2018) tackle this issue via the ODL (Online Deep Learning) framework, which simultaneously learns both neural network parameters and architecture "hedge" weights. ODL is based on *Hedge Backpropagation* (Freund and Schapire, 1997), which alternates between standard backpropagation and a hedge optimization step that adjusts the scalar weights connecting early exit points of the deep neural network. More recently, Agarwal et al. (2023) extended ODL to address the problem of unreliable features in data streams. While these methods partially mitigate the challenges of online neural network training, they continue to face computational complexity and stability issues due to their dependence on hedge backpropagation. At a higher level of abstraction, hedge backpropagation can be viewed as a joint architecture learning task with two interdependent optimization objectives—hedge weights and neural network weights—which can interfere with one another, ultimately slowing the learning process.

Contributions. To overcome these inefficiencies, we present the Multilearner Online Deep Learning (MODL) framework. In contrast to prior approaches, MODL removes hedge backpropagation, a common feature in recent state-of-the-art architectures, and advances the field by: (i) introducing a novel, faster overall framework for jointly learning neural network parameters and topology, and (ii) employing efficient statistical approximations to accelerate learning in a subset of learners via closed-form recursive updates. In summary, our contributions are threefold:

- We propose MODL, a novel framework for online

† Contact: antonios.valkanas@mail.mcgill.ca
‡ This work does not relate to the author's position at Amazon.
⋆ ILLS: International Laboratory on Learning Systems, Mila - Quebec AI Institute

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

deep learning that employs a stacking architecture and hybridizes backpropagation with closed-form updates to accelerate the online training process;

- We derive a fast recursive logistic regression algorithm capable of learning from data streams;
- MODL attains state-of-the-art convergence speed and accuracy on standard benchmarks.

2 RELATED WORK

Multiple learners. Using multiple learners is a well-established idea in machine learning. Two key advantages of using multiple learners are improved point prediction and uncertainty quantification (Lakshminarayanan et al., 2017). Ensemble methods rely on the predictions of multiple models (ensemble members) to produce an overall prediction. Perhaps the most simple way to aggregate the model predictions is a straightforward model average (Dietterich, 2000). Similarly, ensembles can provide uncertainty estimates by calculating the variance of model predictions to provide a confidence interval. The idea of using many models to prevent overfitting, such as bagging for random forests, has been well known for decades (Hastie et al., 2009). Our online learning problem setting does not suit standard ensemble methods that combine the predictions of independently trained base learners trained with the full batch. While there are strategies such as (Shui et al., 2018; D'Angelo and Fortuin, 2021; Masegosa, 2020) for jointly training the constituent members of an ensemble, they do not address online learning and are not trivial to adapt.

Another related area is model cascading (Weiss and Taskar, 2010). Most standard cascading models do not address online learning (Chen et al., 2012). A notable work on cascade learning that works online is from Nie et al. (2024). This work views model cascading as a way to select the minimally expensive model, from a set of large language models, that can answer a natural language query. However, the aim of the work is learning to select the minimal cost model and using that model alone to predict the output. Our work focuses on using the predictions of all models and learning how to combine them effectively with the proposed architecture.

Online learning. The field of online learning has evolved significantly over the past three decades, with early statistical methods tracing back to foundational work by Bottou (1998). A pivotal finding by Bottou and LeCun (2003) demonstrated that online learning algorithms can achieve learning efficiency superior to traditional full batch methods. However, this result is hard to achieve in practice. This revelation laid the groundwork for the widespread adoption of first-order algorithms that aim to learn efficiently via gradient descent. These methods are favored due to their simplicity and low cost (Zinkevich, 2003; Bartlett et al., 2007). A drawback of gradient approaches is that it can be tricky to select the correct learning rate for the algorithm, since there is no validation data available, due to the online nature of the learning task.

Algorithms that employ filtering style updates, such as online generalized linear models (GLMs), can set their own learning rates. One such GLM is online logistic regression, and this has been explored in recent efforts that focused on online optimization. Agarwal et al. (2022) proposed an iterative optimization scheme that regrettably lacks the closed form updates that are needed to improve efficiency. de Vilmarest and Wintenberger (2021) use an extended Kalman filter framework. In this paper we derive online logistic regression directly from the Bayesian approximation of the posterior and also extend the online filtering results to multinomial logistic regression. Our online logistic regression utilizes inherently fast first-order updates.

While second-order methods have been explored in online learning (Hazan et al., 2007; Dredze et al., 2008) and can accelerate parameter convergence, they often come with significant computational overhead due to costly Hessian calculations. To reconcile the speed of the first-order methods with the sample efficiency of the second-order methods, Sahoo et al. (2018) proposed the hedge-backpropagation update. This approach enables simultaneous optimization of both the neural network architecture and parameters, providing a mechanism for dynamic online adjustment of model depth. Building on this work, Agarwal et al. (2023) proposed a scalable method that addresses another key challenge in streaming data: feature reliability. Previously, the issue of unreliable features had only been tackled within the scope of traditional, non-deep learning approaches (Beyazit et al., 2019; He et al., 2019). While Agarwal et al. (2023) propose a first step to handling haphazard feature, they still assume that some base features are always available.

Our approach moves the field forward by departing from hedge backpropagation updates, which have formed the basis of state-of-the-art approaches for the past few years. In this paper, we argue that the use of hedge backpropagation is suboptimal from a learning speed and performance perspective. Instead, we propose a new direction based on a stacking framework that co-trains multiple models. To handle unreliable features we propose a set learner that accepts a variable number of inputs. This eliminates the need for deterministic dropout (Agarwal et al., 2023) and streamlines the gradient flow. An additional, advantage of our set learner approach is that we eliminate the assumption

of always available base features.

Motivation & Applications

Online learning has three main application areas. First, it is useful in situations where retaining training data is undesirable due to privacy concerns or legal constraints. To reduce the risk of data breaches, sensitive information can be utilized for training without being stored (Yang et al., 2022), necessitating the ability to train models with a single pass over the data (Min et al., 2022). Some data are ephemeral by law. Regulations often mandate that certain financial transaction records (*e.g.*, from online purchases) be retained for only a short duration to protect consumer privacy. Under laws like HIPAA in the United States, patient data must be handled with strict confidentiality, and certain data must be purged or anonymized after specific timeframes, especially for non-essential records.

As a result, training typically begins with a limited dataset, followed by online learning as new data become available (Graas et al., 2023). Second, in data-intensive environments such as IoT devices (Abdel Wahab, 2022), real-time surveillance systems (Zhang et al., 2020), or recommendation systems (Valkanas et al., 2025), storing and training on the entire data stream is impractical, making online learning essential. For example, the enormous volumes of data generated by experiments at CERN[1] are largely discarded after short storage periods, with only minimal amounts retained (et al., 2023). Third, online learning is crucial for adapting models to discrepancies between training and deployment environments, as well as to handle distribution shifts and domain generalization—an important challenge in healthcare machine learning (Ktena et al., 2024). For example, adaptation in medical image segmentation (Valanarasu et al., 2022) and credit card (Mienye and Jere, 2024) or insurance fraud detection (Zhang et al., 2024) frequently requires online training to adjust for the mismatch between training and inference distributions.

3 PROBLEM STATEMENT

We address the Online Learning task with data streams that contain missing features. In this problem setting the data generating process produces a sequence of triplets $\mathcal{T} = \{(\mathbf{z}_1, \mathbf{y}_1, \mathbf{O}_1), \ldots, (\mathbf{z}_T, \mathbf{y}_T, \mathbf{O}_T)\}$ sampled sequentially in T time steps. $\mathcal{T}$ consists of input features $\mathbf{z}_t \in \mathbb{R}^d$, ground truth labels $\mathbf{y}_t \in \mathcal{Y}$ (where $\mathcal{Y}$ has a fixed dimension and may be discrete or continuous depending on the task) and the set of available input indices $\mathbf{O}_t = (\mathbf{o}_j)_t \in \{0,1\}^d$. $\mathbf{O}_t$ is a binary mask encoding observed entries, *i.e.*, it is a vector of missingness indicators such that $(\mathbf{o}_j)_t = 1$ if $\mathbf{z}_t(j)$ is observed, and $(\mathbf{o}_j)_t = 0$ otherwise. During training we do not have access to $\mathcal{T}$; we can only observe an incomplete dataset $\mathcal{D}$. Denoting "not available" by NA, we have:

$$\mathbf{x}_t = \mathbf{z}_t \odot \mathbf{O}_t + \texttt{NA} \odot (\mathbf{1}_d - \mathbf{O}_t), \tag{1}$$

$$\mathcal{D} = [(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_T, \mathbf{y}_T)]\,. \tag{2}$$

Since the problem setting is online learning, the goal is to train a new model from scratch in a streaming dataset setup such that at time t, we have access to only the input features $\mathbf{x}_t$,the mask $\mathbf{O}_t$, and the previous model. The input to the model is a concatenated vector that consists of $\mathbf{x}_t$ and $\mathbf{O}_t$ and has length $2d$. After a prediction $\widehat{\mathbf{y}}_t$ is made, we obtain the output labels $\mathbf{y}_t$. We do not have access to any previous training examples so each datapoint is used for training only once. Online learning models are evaluated by the cumulative predictive error across all time steps $E_{\text{TOTAL}} = \sum_t \epsilon(\mathbf{y}_t, \widehat{\mathbf{y}}_t)$. In general, $\epsilon(\cdot,\cdot)$ is a non-negative valued cost function.

4 METHODOLOGY

The main drawback of neural networks in the online setting is their sensitivity to hyperparameters. For example, it can be very difficult to know the optimal width or learning rate of the network before the dataset has been streamed. Deep neural networks require low learning rates to be stable during training, but this entails slow learning and accumulating large errors for millions of optimization steps. On the other hand, fast learners are typically shallow and train quicker, but lack the ability to learn complex representations. To address the hyperparameter issue as well as learner complexity and learning speed trade-off, we propose a new foundational online learning architecture called MODL: Multilearner Online Deep Learning; this is the main contribution of this paper. This architecture compensates for the limitations of neural learners in the online setting by employing multiple learners along different points of the complexity and learning speed trade-off and with different hyperparameters. Our architecture then automatically selects the best weighted combination of these learner's outputs to minimize the distributional divergence between the predicted distribution and the true data distribution.

4.1 Multilearner Online Deep Learning

Consider a probability space $(\Omega, \mathcal{A}, \mathcal{P})$ with sample space Ω, sigma algebra $\mathcal{A}$ of subsets of Ω, and a convex class probability measure $\mathcal{P}$ on Ω. We define a scoring function $S : \mathcal{P} \times \Omega \mapsto \overline{\mathbb{R}}$, where $\overline{\mathbb{R}}$ is the extended real line. We only score forecasts $P \in \mathcal{P}$ that are integrable for $\omega \in \Omega$, denoting the score

[1] https://home.cern/science/computing/storage

as $S(P, \omega)$. To compare two probabilistic forecasts P, Q, we compute $S(P, Q) = \int S(P, \omega)\, dQ(\omega)$. Scoring rules induce divergence metrics $d(P, Q) : \mathcal{P} \times \mathcal{P} \mapsto (0, \infty) \stackrel{\text{def}}{=} S(Q, Q) - S(P, Q)$. A popular score in the literature that we also adopt in this paper is the logarithmic score, yielding $d(p, q) = \mathrm{KL}(q, p)$, where KL is the Kullback-Leibler divergence (Yao et al., 2018, 2024). Having defined the score, we may now define the overall stacking objective as an optimization. Consider the collection of K candidate models $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_K\}$ that produce K predictive distributions $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K\}$. For each model we generate a prediction $\widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i)$, where θ_i are the model parameters, and assign weight w_i to this prediction. The predicted density is

$$\widehat{p}(\mathbf{y}_t|\mathbf{x}_t) = \sum_{i=1}^{K} w_i\, \widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i). \tag{3}$$

We may obtain the stacking weights by following optimization:

$$\min_{w} \quad d_{\mathrm{KL}}\left(\sum_{i=1}^{K} w_i\, \widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i), p(\mathbf{y}_t|\mathbf{x}_t)\right) \tag{4}$$

$$\text{subject to } \sum_{i=1}^{K} w_i = 1, \quad w_i \geq 0, \tag{5}$$

where $p(\mathbf{y}_t|\mathbf{x}_t)$ is the true distribution. In practice, we optimize over an empirical approximation by sampling the distribution for T time steps yielding the objective:

$$\min_{w} \quad \sum_{t=1}^{T} d_{\mathrm{KL}}\left(\sum_{i=1}^{K} w_i\, \widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i), p(\mathbf{y}_t|\mathbf{x}_t)\right). \tag{6}$$

Since it is standard to use negative log-likelihood (NLL) to train the neural parameters we jointly optimize over w_i as well as the parameters θ_i. We modify the previous optimization due to equivalence of minimizing KL divergence and NLL:

$$\min_{w,\theta} -\sum_{t=1}^{T} \log\left(\sum_{i=1}^{K} w_i \widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i)\right). \tag{7}$$

We interpret $\widetilde{p}_i = w_i \widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i)$ as the latent class score per model i. The latent class scores can be interpreted as an unnormalized/improper probability measure. Note that each neural model directly outputs the weighted prediction $\widetilde{p}_i$ which is not a valid probability distribution in general. The optimization in eq. (7) requires w_i to sum to 1. Since this is not the case as the neural network output is only constrained to be positive we apply softmax to the sum to project to a valid probability distribution as shown in Figure 1. This architectural decision differentiates our approach from ensembling as we do not aggregate predicted class probabilities and we do not train models independently of each other. Instead, the latent class scores are aggregated via sum pooling and then passed through one softmax layer to produce the weighted final prediction. Thus the i-th model f_i, $f_i(\mathbf{x}_t, \theta_i) \equiv w_i \widehat{p}(\mathbf{y}_t|\mathbf{x}_t, \mathcal{M}_i, \theta_i)$, learns both the associated probability distribution and its weight, implicitly, while the final prediction has the following form:

$$\widehat{p}(\mathbf{y}_t|\mathbf{x}_t) = \mathrm{softmax}\left[\sum_{i=1}^{K} f_i(\mathbf{x}_t, \theta_i)\right]. \tag{8}$$

We demonstrate in our experiments that the proposed modeling approach enables mixing of the learners with heterogeneous learning paradigms (e.g. backpropagation and approximate closed-form) and is more sample effective compared to the procedures that attempt learning both weights and architectures explicitly (e.g. hedge backpropagation based Aux-Drop or ODL). Since the scores $f_i(\cdot)$ are not probabilities, they are unbounded (besides being non-negative), therefore, models that are confident in their prediction assign larger scores for a particular class. This is an implicit and learnable way for each model to select its own confidence level.

Selecting the learners The learners, which can be interpreted as functional priors, must be selected by the user before training. One could argue that this is a limitation as depending on the group of learners we select MODL can perform differently. We, however, argue that this is a good thing as it forces the user to explicitly state their assumptions. By selecting only deep models the user assumes that the distribution is too difficult to learn using a linear learner. Conversely, by selecting many models along various points of the bias-variance trade off, the user demonstrates ignorance of the complexity of the joint feature-label distribution. In practice, to select the group of learners we propose the following workflow: (i) select powerful generic architectures that are applicable to the problem at hand; (ii) define the *smallest* and the *largest* capacity models that would be plausibly expected to learn the data distribution; (iii) define several models in-between the largest and the smallest capacity models; (iv) train the aforementioned models under the MODL framework.

In the rest of the methodology section, we propose a fast learner and a slow but powerful set learner, along with a standard MLP that serves as a medium learner. We employ these learners in all our experiments with universally good results. Here, the fast learner based on online logistic regression quickly learns a linear approximation to the solution and provides the strongest signal guiding the global model output

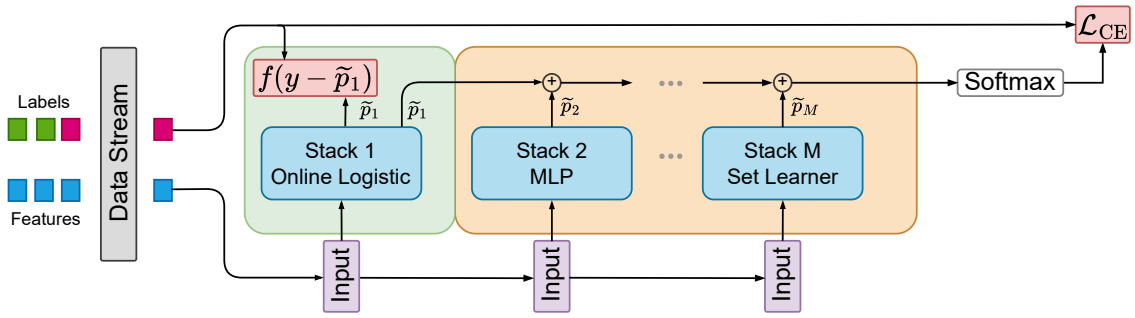

Figure 1: Multilearner Online Deep Learning. The dataset is streamed sequentially. Fast learners quickly adapt to the data distribution, providing a strong baseline for the deeper models. By synergizing models with different bias-variance trade-offs, the overall architecture quickly adapts to the data and learns deep representations. Individual model latent class scores $\widetilde{p}_i$ are sum pooled, then projected into class probabilities. During co-learning models learn to predict on top of each other. Non-neural learners (green box) learn via filtering style updates; $f(y-\widetilde{p}_1)$ is eq. (11). Neural learners (orange box) learn via backpropagation of cross entropy loss $\mathcal{L}_{\text{CE}}$.

initially. As more data arrive, the second stage learner, implemented as an MLP, kicks in, becoming the most accurate learner before the heavier model has had the time to converge sufficiently. Finally, the set learner, which has a deep structure and the ability to represent the semantics of variables, provides the modeling output that is the finest and therefore the hardest to learn. For example, learning semantic embeddings of variable IDs may take a long time, but this endows this part of the architecture with the ability to handle missing data and encode complex input-output relationships. Of course, this can be generalized beyond the three aforementioned learning levels. We choose to outline the structure used in our experiments, which also happens to capture the most important methodological thinking of our approach.

Fast Learner. Consider the standard task of fitting a logistic regression model. Denote the dataset by $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with $\mathbf{x} \in \mathbb{R}^m, y_i \in \{0, 1\}$. For a simple generalized linear model $p(y|\mathbf{x}, \theta) = \sigma(\mathbf{x}\theta)$, with weights $\theta \in \mathbb{R}^m$, where $\sigma(\cdot)$ is the logistic function, we assign a normal prior $p(\theta) = \mathcal{N}(\theta; \mathbf{m}_0, \mathbf{P}_0)$, with mean $\mathbf{m}_0 \in \mathbb{R}^m$ and symmetric positive definite covariance matrix $\mathbf{P}_0 \in \mathbb{R}^{m\times m}$. Proposition 1 derives a closed form filtering style update for the model parameters for each observation.

Proposition 1. *Assuming an input feature distribution for* $\mathbf{x}$ *that is approximately normal, and linearizing the non-linear relationship,* $\mathbf{y} = \sigma(\theta\mathbf{x})$*, a quadratic approximation to the posterior of model weights after observing the n-th datapoint is given by the recursive formula* $p(\theta|\{(\mathbf{x}_i, y_i)\}_{i=1}^n) \approx \mathcal{N}(\theta; \mathbf{m}_n, \mathbf{P}_n)$*. For logistic regression we take* $\sigma(\cdot)$ *to be the logistic function.*

The proof, concrete formulae yielding $\mathbf{m}_n, \mathbf{P}_n$, and multinomial extension are provided in Appendix K.

Algorithm 1 Online Bayesian Logistic Regression

Require: Dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
Initialize: $p(\theta) = \mathcal{N}(\theta; \mathbf{m}_0, \mathbf{P}_0)$, $\mathbf{m}_0 = 0$, where $\mathbf{P_0} = \mathbf{I}$
Return $p(\theta|\{(\mathbf{x}_i, y_i)\}_{i=1}^n) = \mathcal{N}(\theta; \mathbf{m}_n, \mathbf{P}_n)$
for t = 1,. . . ,n **do**
Receive instance: $\mathbf{x}_t$ and predict: $\hat{y}_t = \sigma(\mathbf{x}_t\mathbf{m}_{t-1})$
Reveal true value: y_t and update parameters:

$$\Xi_t = \mathbf{x}_t\mathbf{P}_{t-1}\mathbf{x}_t^\top + \mathbf{P}\left[(1-\sigma(\mathbf{x}_t\mathbf{m}_{t-1}))\,\sigma\left(\mathbf{x}_t\mathbf{m}_{t-1}\right)\right]^2, \tag{9}$$

$$\mathbf{K}_t = \mathbf{P}_{t-1}\mathbf{X}_t^\top\left(1-\sigma(\mathbf{x}_t\mathbf{m}_{t-1})\right)\sigma\left(\mathbf{x}_t\mathbf{m}_{t-1}\right)\Xi_t^{-1}, \tag{10}$$

$$\mathbf{m}_t = \mathbf{m}_{t-1} + \mathbf{K}_t[\mathbf{y}_t - \sigma\left(\mathbf{x}_t\mathbf{m}_{t-1}\right)], \tag{11}$$

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \mathbf{K}_t\Xi_t\mathbf{K}_t^\top. \tag{12}$$

end for

As a consequence of this recursive formula, we derive the sequential Algorithm 1, based on the quadratic approximation to the log-likelihood. This algorithm can process the dataset in one pass, while never storing any data in memory. Note that Ξ_t^{-1} in Algorithm 1 is a scalar, so there is no need to invert any matrices in our approach. The only memory requirement is storage of the parameters of the normal posterior distribution. The parameters update proportionally to the "innovation" (eq. (11)), which is data dependent, rather than set by the user. This fast learner is a high bias and low variance model. Next, we propose a highly expressive module that lies at the opposite end of the

bias-variance spectrum.

Slow learner with set inputs. Rather than masking missing features with a zero or using deterministic dropout, as is done in (Agarwal et al., 2023), we treat the input as a set that excludes any missing features. Recall that the data generating process produces a sequence of triplets $\mathcal{T} = \{(\mathbf{z}_1, \mathbf{y}_1, \mathbf{O}_1), \ldots, (\mathbf{z}_T, \mathbf{y}_T, \mathbf{O}_T)\}$. Then, the set of input features $\mathcal{X}_t$ can be expressed as:

$$\mathcal{X}_t = \{\mathbf{z}_{t,j} : \mathbf{O}_{t,j} = 1\}, \quad \mathcal{I}_t = \{j : \mathbf{O}_{t,j} = 1\}, \quad (13)$$

$$\mathcal{D} = [(\mathcal{X}_1, \mathcal{I}_1, \mathbf{y}_1), \ldots, (\mathcal{X}_T, \mathcal{I}_T, \mathbf{y}_T)]. \quad (14)$$

The size of the input feature set $\mathcal{X}_t$ is time varying. To allow the model to determine which inputs are available, it is necessary to pass a set of feature IDs in an index set $\mathcal{I}_t$. Our proposed set learning module follows closely the ProtoRes architecture (Oreshkin et al., 2022). It takes the index set $\mathcal{I}_t$ and maps each of its active index positions to a continuous representation to create feature ID embeddings. It then concatenates each ID embedding to the corresponding feature value of $\mathcal{X}_t$. These feature values and ID embedding pairs are aggregated and summed to produce fixed dimensional vector representations that we denote as $\mathbf{x}_0$. The main components of our proposed set learning module are blocks, each consisting of L fully connected (FC) layers. Residual skip connections are included in the architecture so that blocks can be bypassed. An input set $\mathcal{X}_t$ is mapped to $\mathbf{x}_{0,t} = \text{EMB}(\mathcal{X}_t, \mathcal{I}_t)$. The overall structure of the module at block $r \in \{1, \ldots, R\}$ (see Fig. 4) is (we are dropping time index for conciseness):

$$\begin{aligned}
\mathbf{h}_{r,1} &= \text{FC}_{r,1}(\mathbf{x}_{r-1}), \ \ldots, \ \mathbf{h}_{r,L} = \text{FC}_{r,L}(\mathbf{h}_{r,L-1}), \\
\mathbf{x}_r &= \text{ReLU}(\mathbf{W}_r \mathbf{x}_{r-1} + \mathbf{h}_{r-1,L}), \\
\hat{\mathbf{y}}_r &= \hat{\mathbf{y}}_{r-1} + \mathbf{Q}_L \mathbf{h}_{r,L},
\end{aligned}$$

where $\mathbf{W}_r, \mathbf{Q}_L$ are learnable matrices. We connect R blocks sequentially to obtain the global output $\hat{\mathbf{y}}_R$.

5 EXPERIMENTS

Our experiments provide empirical support to the following: (i) our online learning model MODL consistently outperforms state-of-the-art online deep learning methods; (ii) new proposed modules, including the online logistic regression as well as the set learning component, work synergistically within the MODL framework; (iii) combining the learners is best done via summation of unnormalized class scores, as opposed to alternatives, such as a mixture of experts or regular ensembles; (iv) our optimization framework is significantly more time efficient than hedge backpropagation based Aux-Drop (ODL) by Agarwal et al. (2023), which is the current best model.

Datasets. We employ established online deep learning benchmarks, replicating the setup from prior work by Agarwal et al. (2023). Our analysis covers eight datasets of varying sizes. As noted in the review section, particle physics colliders generate vast amounts of data, necessitating the ability to learn from streaming data. Consequently, we selected two large particle physics datasets: *HIGGS* and *SUSY* (Baldi et al., 2014). The *HIGGS* dataset focuses on detecting the HIGGS boson in particle accelerator experiments, while *SUSY* aims to identify SUper SYmmetric (SUSY) particles, distinguishing signal from background noise, which is particularly challenging. In addition, we evaluate our model on the *german* dataset (Chang and Lin, 2011), which assesses consumer credit risk. We also include *svmguide3* (Dua and Graff, 2017), a synthetic binary classification dataset. Another physics dataset, *magic04* (Chang and Lin, 2011), focuses on detecting gamma particle radiation in Cherenkov telescope images. *a8a* (Dua and Graff, 2017) uses census data to predict individuals with high incomes based on demographic factors. We also test on standard image classification datasets including *Infinite-MNIST* (I-MNIST) (Bottou et al., 2007) and *CIFAR-10* (Krizhevsky et al., 2012). Details about the datasets can be found in Appendix F.

Baselines. We compare to standard methods from the literature. For experiments with unreliable features the current standard methods are OLVF (Beyazit et al., 2019) and two versions of Aux-Drop (Agarwal et al., 2023). The first version uses a fixed architecture and online gradient descent (ODG) updating (Zinkevich, 2003), and the second uses ODL (Sahoo et al., 2018) hedge updates to learn the architecture jointly with the parameters. The latter variant of Aux-Drop is considered state-of-the-art. For the larger datasets we compare directly with the best method as other baselines are not designed for large datasets and very deep models. Continual learning approaches such as work by Kirkpatrick et al. (2017) or Prabhu et al. (2020) are not applicable as we are training from scratch.

Training Methodology. We closely follow the experimental setup of Agarwal et al. (2023). For small and medium datasets we run 20 independent trials, whereas for the large datasets we run 5. The random input masks for missing features and network initializations are from the same seed to ensure fairness. For all methods at each training step we process a single instance, so the batch size is fixed to 1. Hyperparameter sensitivity analysis is studied in Appendices E.3 and E.4. For baselines we use the best hyperparameters reported in the literature. For missing feature experiments we randomly mask all features with some probability (except for the first two features that are always available). Note that this is done to replicate the setting of prior work for the purpose of comparing

Table 1: Comparison on standard datasets: cumulative error (mean ± st. deviation) over 20 runs. Bold indicates statistically significant result on Wilcoxon test ($p < 0.05$).

Dataset	OLVF	Aux-Drop (ODL)	Aux-Drop (OGD)	MODL (ours)
german	333.4±9.7	306.6 ± 9.1	327.0±45.8	**286.1 ± 5.3**
svmguide3	346.4±11.6	296.9±1.3	296.6±0.6	**288.0 ± 1.0**
magic04	6152.4±54.7	5607.1 ± 235.1	5477.4 ± 299.3	**5124.7 ± 153.4**
a8a	8993.8±40.3	6700.4±124.5	7261.8±283.5	**5670.5 ± 278.2**

Table 2: Comparison on HIGGS and SUSY for various feature probabilities p_f. Here, p_f represents the i.i.d probability of a given feature being available during a time step. The metric is the mean (± st. deviation) cumulative error in thousands over 5 runs. Bold indicates statistically significant Wilcoxon test ($p < 0.05$).

p_f	HIGGS	
	Aux-Drop (ODL)	MODL (ours)
.01	440.2 ± 0.1	**439.6 ± 0.1**
.20	438.4 ± 0.1	**435.6 ± 0.4**
.50	427.4 ± 0.7	**422.7 ± 0.3**
.80	411.8 ± 0.4	**399.6 ± 0.2**
.95	399.4 ± 1.0	**377.1 ± 0.5**
.99	392.0 ± 1.0	**366.4 ± 0.5**

p_f	SUSY	
	Aux-Drop (ODL)	MODL (ours)
.01	285.0 ± 0.1	**283.0 ± 0.1**
.20	274.8 ± 0.9	**271.8 ± 0.1**
.50	256.6 ± 1.0	**252.0 ± 0.1**
.80	237.0 ± 0.7	**230.6 ± 0.1**
.95	226.2 ± 0.4	**217.5 ± 0.1**
.99	222.3 ± 0.2	**212.2 ± 0.2**

Table 3: Multiclass experiments missclassification rate. MODL outperforms both on small (CIFAR-10, 50k samples) and large datasets (I-MNIST, 1M samples).

Experiment	Aux-Drop (ODL)	MODL
CIFAR-10	88.9±0.3%	**66.1±0.1**%
I-MNIST 1M	8.2±0.2%	**3.8±0.1**%

the algorithms; our algorithm does not require any features to be always available.

MODL Architecture: We used the exact same architecture in all experiments. Specifically, the online logistic regression learner is implemented identically for all datasets and the MLP has 3 hidden layers with 250 neurons. The set learner consists of 6 blocks with 3 layers per block. The width of each layer is set to 128 neurons. For example, CIFAR-10 the capacity of the online logistic, MLP and Set learners is 3072, 893750, 1447312 learnable parameters respectively. For a detailed description of hyperparameters, see Appendix H.

Key Results. Figure 2 shows that during training MODL remains consistently ahead of the current state-of-the-art model Aux-Drop (ODL) with respect to the classification error rate. Besides converging faster, our approach also achieves a lower overall error rate at the end of training. These findings are replicated across datasets that vary in size over 3 orders of magnitude and with different feature missingness levels. The detailed results for the cumulative miss-classification metric are summarized in Tables 1 and 2. For the small and medium datasets shown in Table 1 we can see that MODL is the clear top performer, reducing error on average by 11%. We note that the improvements are measured over 20 trials and are all statistically significant at the 0.05 level using a paired Wilcoxon test. For the large datasets in Table 8 we conduct a detailed feature missingness experiment for 13 different noise levels with 5 trials per level. The results show that MODL is consistently improving the state-of-the-art for all noise levels. This is an important result as it verifies the strength of our approach for a wide spectrum of feature noise, from near-zero up to extreme levels. We also validate our approach on multiclass image problems including the I-MNIST dataset with one million images and CIFAR-10 in Table 3 where we achieve top performance. Note that on the complex but small CIFAR-10 dataset, Aux-Drop (ODL) struggles to learn anything in 50,000 steps (size of the dataset) as it has to both learn the architecture and the parameters of the early exit classifiers. On the other hand MODL quickly converges to reasonable performance. This is also reflected in large scale experiments in Figure 2 such as HIGGS. Here, MODL has reasonable performance at 50,000 steps whereas Aux-Drop (ODL) has barely improved over untrained performance.

The learning efficiency of MODL optimization compared to Aux-Drop (ODL) extends past faster convergence. Aux-Drop (ODL), and base ODL operate at a higher training time complexity. The time complexity of performing a backpropagation update for a network with L layers is $O(L^2)$ for standard implementations of ODL and Aux-Drop (ODL). This is analyzed in Appendix C, where it is also empirically verified. On the other hand, backpropagation updates in MODL cost $O(L)$. In our large scale experiments, compared to the

Table 4: Ablation study on merging constituent learner outputs. We compare our proposed score sum approach to (i) mixture of experts (MoE); (ii) multiplying the learner logit probabilities; (iii) ensemble learning; (iv) an ad-hoc greedy weighting scheme that assigns more weight to learners with higher sliding window accuracy. HIGGS (p_f=0.5), SUSY (p_f=0.99); p_f represents the i.i.d probability of a given feature being available during a time step.

Dataset	Greedy Weighing	Multiplication	Ensemble	Mix. of Experts	Score Sum (ours)
german	293.0±10.3	294.9±7.1	307.5±21.0	316.2±9.3	**285.9±7.2**
svmguide3	296.3±1.5	299.5±6.5	296.6±1.5	298.4±3.3	**287.5±4.0**
magic04	**4720±154**	5146±75	6506± 588	6719±44	5226 ± 98
a8a	6495±709	5691±40	5865±40	6190±168	**5673±36**
HIGGS	442.8±0.3k	422.7±0.6k	428.8±0.1k	431.9±0.9k	**422.7±0.3k**
SUSY	220.2±0.8k	212.3±0.1k	218.7±0.2k	217.0±0.2k	**212.2±0.2k**

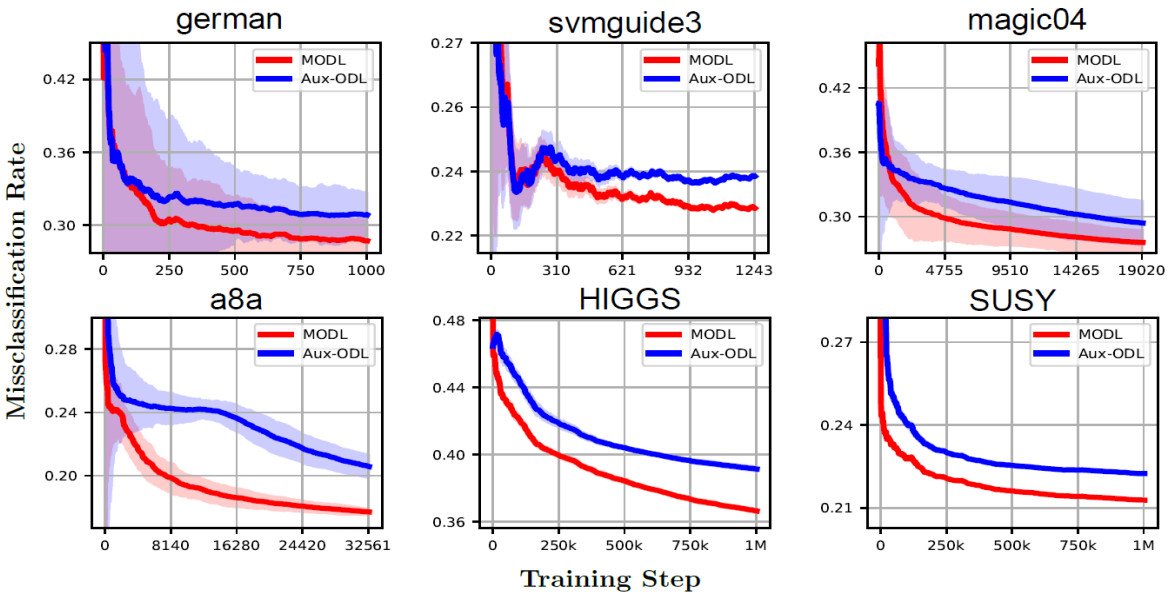

Figure 2: Comparison of missclassification rate (lower is better) as a function of online training step for Aux-Drop (ODL) (Agarwal et al., 2023) (blue) vs. our proposed model MODL (red). Shaded regions indicate 95% C.I.

AuxDrop (ODL) algorithm, our proposed MODL optimization reduces the the training time dramatically, *e.g.*, from 63 hours to 9 when training with MLPs with the same number of layers and embedding dimension.

Validation of MODL latent score summing. The second ablation experiment explores different ways to merge learner predictions. MODL merges predictions by direct summation of the constituent learners' latent class scores. This is an important ablation result as it validates the MODL architecture and shows that letting the learners weigh their prediction individually works better than standard approaches. We consider alternative approaches that include: (i) learnable gating functions in a Mixture of Experts (MoE) style setup where the final output is a weighted sum of each learner's prediction; (ii) multiplying the logit probabilities (by summing the logarithm of the predicted logits); (iii) regular ensemble learning; and (iv) a greedy weighting scheme that assigns more weight to learners with higher sliding window accuracy. Our results in Table 4 demonstrate the strength of latent score aggregation as an effective co-learning approach in MODL. Baseline implementation details are available in Appendix G.

Ablation Studies. In this section we rigorously verify our model components. Table 5 shows that removing any individual component negatively affects the overall performance. From these results we ascertain that the presence of diverse learners with respect to the bias-variance trade-off bolsters performance. Additionally, in Table 6 we provide the results of an ablation study that investigates the use of online closed form updates versus a standard gradient based updating scheme for the logistic regression learner. Here, G-LR (gradient-based logistic regression) is the control model where the closed form solution is replaced by back-propogation. We keep all parameters the same as in

Table 5: Ablation of proposed model components. We show that for each learner that we add performance improves. The best model is our proposed MODL which uses all 3 components: online logistic regression (OLR), MLP and a set learner; p_{f} represents the i.i.d probability of a given feature being available during a time step.

$p_{\mathbf{f}}$	HIGGS			SUSY		
	OLR + MLP	OLR + Set	MODL (ours)	OLR + MLP	OLR + Set	MODL (ours)
.01	442.7±0.2	450.2±0.4	**439.6 ± 0.1**	285.3±0.1	332.6±0.3	**283.0 ± 0.1**
.20	439.9±0.1	450.3±0.2	**435.6 ± 0.4**	274.0±0.1	352.4±0.3	**271.8 ± 0.1**
.50	430.3±0.2	450.1±0.1	**422.8 ± 0.3**	255.0±0.1	343.3±0.6	**252.0 ± 0.1**
.80	411.7±0.1	449.6±0.2	**399.6 ± 0.2**	234.3±0.1	326.3±0.4	**230.6 ± 0.1**
.95	394.1±0.5	448.5±0.1	**377.1 ± 0.5**	222.1±0.1	320.9±0.4	**217.5 ± 0.1**
.99	386.6±0.1	447.1±0.1	**366.4 ± 0.6**	217.3±0.1	316.3±0.6	**212.2 ± 0.2**

Table 6: Ablation of proposed online update scheme for the logistic regression learner. G-LR (gradient-based logistic regression) is the control model where the closed form solution is replaced by backpropogation. We conduct 100 and 25 runs for the small and large datasets respectively.

Dataset	MODL (G-LR)	MODL (ours)
german	325.2 ± 8.9	**289.4 ±5.4**
svmguide3	293.0± 3.3	**287.5 ±3.6**
magic04	5652.5 ±51.9	**5137.1 ±53.5**
a8a	5829.6 ±27.1	**5493.2 ±51.2**
HIGGS	384.4 ±0.2	**362.3 ±0.2**
SUSY	214.0 ±0.1	**210.5 ±0.2**

the original experiment. We set the learning rate for the gradient-based logistic regression updates to 0.01 (small datasets) and 0.001 (larger datasets). These values are the same leaning rates as used for the other gradient-based algorithms in MODL. Comparing the closed form solution MODL (ours) with the gradient-based solution MODL (G-LR) demonstrates the benefit of the closed form solution.

Robustness to hyperparameter selection. A particularly challenging and sensitive hyperparameter to select in online learning is the learning rate. We run sensitivity experiments with respect to the learning rate in the large benchmarks. Our results in Appendix E.4 show increased robustness to learning rate selection. This is expected because our framework incorporates learners that train without backpropagation, which safeguards against selecting learning rates that are far from optimal. Our framework MODL may work with as many learners as the user wishes to implement. In the main experiments we proposed a standard setup that achieves practically good results with 3 learners. However, this does not preclude the user from introducing more than 3 learners. In the next experiment we implement a 5 learner setup by introducing MLPs with additional layers to obtain learners with different characteristics along the bias-variance tradeoff. We investigate the robustness of MODL to the number of learners in Table 7.

Table 7: MODL demonstrates low sensitivity to number of learners selected, showing robustness to hyperparameter selection. We observe that MODL performance is stable when we use three or five learners. The two additional learners are higher capacity MLPs.

Dataset	3 models MODL	5 models MODL
german	286.1 ± 5.3	290.1 ± 7.3
svmguide3	288.0 ± 1.0	287.0 ± 4.1
magic04	5124.7 ± 153.4	5239.7 ± 175.2
a8a	5670.5 ± 278.2	5639.3 ± 144.2

6 CONCLUSION

Limitations. Our work is focused on supervised learning from streams of data. This is not trivially applicable to other areas such as online reinforcement learning. While our work shows that including multiple learners with different bias-variance trade-offs is a sensible approach to online deep learning, it does not come with theoretical guarantees for error or regret bounds.

Overview. We demonstrate that the problem of online deep learning is best handled synergistically by multiple learners that operate at various points of the convergence speed/expressivity trade-off curve. We derive a very fast learning and non-backpropagation based approach to support early stages of learning and to provide a strong baseline for other learners. Then, at the other end of the spectrum, we propose a slowly learning, yet very expressive set learner that is invariant to feature ordering and robust to missing features. We show that the appropriate way to combine these learners is a novel stacking regime where the models learn to predict on top of one-another rather than ensembling or via a mixture of experts. Our approach significantly improves performance and reduces training complexity.

Acknowledgements

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), [reference number 260250]. Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence 260250]. During this research project Antonios Valkanas was supported through NSERC Postgraduate Scholarships – Doctoral (PGS-D) program, the Stavros S. Niarchos Foundation Fellowship and the Vadasz Doctoral Fellowship.

References

[1] Omar Abdel Wahab. Intrusion detection in the IoT under data and concept drifts: Online deep learning approach. *IEEE J. Internet of Things*, 9 (20):19706–19716, 2022.

[2] Naman Agarwal, Satyen Kale, and Julian Zimmert. Efficient methods for online multiclass logistic regression. In *Proc. Int. Conf. Alg. Learning Theory (ALT)*, pages 3–33, Paris, France, Mar. 2022.

[3] Rohit Agarwal, Deepak Gupta, Alexander Horsch, and Dilip K. Prasad. Aux-drop: Handling haphazard inputs in online learning using auxiliary dropouts. *Trans. Mach. Learn. Research (TMLR)*, 2023.

[4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Commun.*, 5, 2014.

[5] Peter L. Bartlett, Elad Hazan, and Alexander Rakhlin. Adaptive online gradient descent. In *Proc. Neural Info. Proces. Sys. (NeurIPS)*, page 65–72, Red Hook, NY, USA, Dec. 2007.

[6] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Mach. Learn. Res.*, 7:2399–2434, 2006.

[7] Ege Beyazit, Jeevithan Alagurajah, and Xindong Wu. Online learning from data streams with varying feature spaces. In *Proc. Conf. Artificial Intell. (AAAI)*, pages 3232–3239, Feb. 2019.

[8] Dankmar Böhning. Multinomial logistic regression algorithm. *Annals of the Institute of Statistical Mathematics*, 44:197–200, 1992.

[9] Léon Bottou. *Online Learning and Stochastic Approximations*, chapter 2, pages 9–42. Cambridge University Press, 1998.

[10] Léon Bottou and Yann LeCun. Large scale online learning. In *Proc. Adv. Neural Info. Proces. Sys. (NeurIPS)*, pages 217–224, Vancouver, Canada, Dec. 2003.

[11] Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston. *Training Invariant SVMs Using Selective Sampling*, pages 301–320. MIT Press, 2007.

[12] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), May 2011.

[13] Minmin Chen, Zhixiang Xu, Kilian Weinberger, Olivier Chapelle, and Dor Kedem. Classifier cascade for minimizing feature evaluation cost. In *Proc. Int. Conf. Artificial Intell. Stat. (AISTATS)*, volume 9, pages 218–226, Canary Islands, Spain, May 2012.

[14] Merlise Clydec and Edwin S Iversen. *Bayesian model averaging in the M-open framework*. Oxford University Press, 2013.

[15] Francesco D'Angelo and Vincent Fortuin. Repulsive deep ensembles are bayesian. In *Proc. Int. Conf. Neural Info. Proces. Syst. (NeurIPS)*, pages 3451–3465, Online, Dec. 2021.

[16] Joseph de Vilmarest and Olivier Wintenberger. Stochastic online optimization using kalman recursion. *J. Machine Learning Research*, 22(223):1–55, 2021.

[17] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proc. Int. Workshop on Multiple Classifier Systems*, page 1–15, Cagliari, Italy, Jun. 2000.

[18] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proc. Int. Conf. Machine Learning (ICML)*, page 264–271, Helsinki, Finland, Jul. 2008.

[19] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[20] T. Basaglia et al. Data preservation in high energy physics. *Euro. Phys. J. C*, 83:1–41, 2023.

[21] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comp. Sys. Sci.*, 55(1): 119–139, 1997.

[22] Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *J. Amer. Statistical Assoc.*, 102(477):359–378, 2007.

[23] Adriaan B.M. Graas, Sophia Bethany Coban, Kees Joost Batenburg, and Felix Lucka. Just-in-time deep learning for real-time x-ray computed tomography. *Scientific Reports*, 13, 2023.

[24] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In

Proc. Foundations Comp. Science, (FOCS), pages 359–366, Redondo Beach, CA, USA, Nov. 2000.

[25] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[26] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Mach. Learn.*, 69(2-3):169–192, 2007.

[27] Yi He, Baijun Wu, Di Wu, Ege Beyazit, Sheng Chen, and Xindong Wu. Online learning from capricious data streams: A generative approach. In *Proc. Int. Joint Conf. on Artificial Intell. (IJCAI)*, pages 2491–2497, Macao, China, Aug. 2019.

[28] Steven C. H. Hoi, Rong Jin, Peilin Zhao, and Tianbao Yang. Online multiple kernel classification. *Mach. Learn.*, 90(2):289–316, 2013.

[29] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proc. national academy of sciences*, 114 (13):3521–3526, 2017.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Adv. Neural Info. Proc. Sys. (NeurIPS)*, pages 84–90, Vancouver, Canada, Dec. 2012.

[31] Ira Ktena, Olivia Wiles, Isabela Albuquerque, Sylvestre-Alvise Rebuffi, Ryutaro Tanno, Abhijit Guha Roy, Shekoofeh Azizi, Danielle Belgrave, Pushmeet Kohli, Taylan Cemgil, et al. Generative models improve fairness of medical classifiers under distribution shifts. *Nature Medicine*, pages 1–8, 2024.

[32] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proc. Int. Conf. Neural Info. Proces. Syst. (NIPS)*, page 6405–6416, Long Beach, CA, USA, Dec. 2017.

[33] Tri Le and Bertrand Clarke. A Bayes Interpretation of Stacking for $\mathcal{M}$-Complete and $\mathcal{M}$-Open Settings. *Bayesian Analysis*, 12(3):807 – 829, 2017.

[34] Andrés R. Masegosa. Learning under model misspecification: applications to variational and ensemble methods. In *Proc. Int. Conf. Neural Info. Proces. Sys. (NeurIPS)*, pages 5479–5491, Vancouver, Canada, Dec. 2020.

[35] Ibomoiye Domor Mienye and Nobert Jere. Deep learning for credit card fraud detection: A review of algorithms, challenges, and solutions. *IEEE Access*, 12:96893–96910, 2024.

[36] Youngjae Min, Kwangjun Ahn, and Navid Azizan. One-pass learning via bridging orthogonal gradient descent and recursive least-squares. In *Proc. IEEE Conf. Decision & Control (CDC)*, pages 4720–4725, Cancun, Mexico, Dec. 2022.

[37] Lunyiu Nie, Zhimin Ding, Erdong Hu, Christopher Jermaine, and Swarat Chaudhuri. Online cascade learning for efficient inference over streams. In *Proc. Int. Conf. Machine Learning (ICML)*, page 38071–38090, Jul. 2024.

[38] Boris N. Oreshkin, Florent Bocquelet, Félix G. Harvey, Bay Raitt, and Dominic Laflamme. Protores: Proto-residual network for pose authoring via learned inverse kinematics. In *Proc. Int. Conf. Learning Representations (ICLR)*, Online, Apr. 2022.

[39] Ameya Prabhu, Philip Torr, and Puneet Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European Conf. Computer Vision (ECCV)*, Online, Aug. 2020.

[40] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: learning deep neural networks on the fly. In *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, page 2660–2666, Stockholm, Sweden, Jul. 2018.

[41] Changjian Shui, Azadeh Sadat Mozafari, Jonathan Marek, Ihsen Hedhli, and Christian Gagné. Diversity regularization in deep ensembles. *arXiv preprint arXiv:1802.07881*, 2018.

[42] Jeya Maria Jose Valanarasu, Pengfei Guo, VS Vibashan, and Vishal M. Patel. On-the-fly test-time adaptation for medical image segmentation. In *Int. Conf. Medical Imaging with Deep Learning*, Jul. 2022.

[43] Antonios Valkanas, Yuening Wang, Yingxue Zhang, and Mark Coates. Personalized negative reservoir for incremental learning in recommender systems. *Trans. Mach. Learn. Research (TMLR)*, 2025.

[44] Larry Wasserman. Bayesian model selection and model averaging. *J. Math. Psychol.*, 44(1):92–107, Mar. 2000.

[45] David Weiss and Benjamin Taskar. Structured prediction cascades. In *Proc. Int. Conf. Artificial Intell. Stat. (AISTATS)*, volume 9, pages 916–923, Chia, Italy, May 2010.

[46] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Proc. Adv. Neural Info. Proc. Sys. (NeurIPS)*, pages 4697–4708, Vancouver, Canada, Dec. 2020.

[47] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[48] Luyu Yang, Mingfei Gao, Zeyuan Chen, Ran Xu, Abhinav Shrivastava, and Chetan Ramaiah. Burn after reading: Online adaptation for cross-domain streaming data. In *Proc. Euro. Conf. Comp. Vision (ECCV)*, page 404–422, Tel Aviv, Israel, Oct. 2022.

[49] Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. Using stacking to average bayesian predictive distributions (with discussion). *Bayesian Analysis*, 13(3):917–1003, 2018.

[50] Yuling Yao, Bruno Régaldo-Saint Blancard, and Justin Domke. Simulation-based stacking. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *Proc. Int. Conf. Artificial Intell. Stat. (AISTATS)*, pages 4267–4275, Valencia, Spain, May 2024.

[51] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. Onrl: improving mobile video telephony via online reinforcement learning. In *Proc. Int. Conf. Mobile Comp. Networking*, pages 1–14, London, United Kingdom, Sep. 2020.

[52] Rui Zhang, Dawei Cheng, Jie Yang, Yi Ouyang, Xian Wu, Yefeng Zheng, and Changjun Jiang. Pretrained online contrastive learning for insurance fraud detection. In *Proc. Conf. Artificial Intell. (AAAI)*, volume 38, pages 22511–22519, Vancouver, Canada, Feb. 2024.

[53] Martin A. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proc. Int. Conf. Machine Learning (ICML)*, pages 928 – 935, Washington, DC, USA, Aug. 2003.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]
   Code will be made public upon paper acceptance.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes/No/Not Applicable]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

MODL: Multilearner Online Deep Learning Supplementary material

Table of Contents

A Broader Impact Statement **15**

B Additional Background **16**
B.1 Model weighing approaches 16
B.2 Online Learning with Missing Features 16

C Fast Online Deep Learning **17**

D Algorithms **18**

E Additional Experiments **19**
E.1 Feature Missingness Experiment 19
E.2 Model Ablation Study 19
E.3 Training Time Additional Settings 20
E.4 Learning Rate Sensitivity Experiments 20

F Dataset Statistics **22**

G Score Sum Validation Implementation Details **22**

H Hyperparameters and Computational Resources **23**
H.1 Hyperparameters 23
H.2 Computational Resources 24

I Bayesian Filters for Online Regression **24**
I.1 De Finetti Representation Theorem 24
I.2 Bayesian Linear Regression 25
I.3 Online Bayesian Linear Regression 25

J Logistic Regression **26**

K Online Logistic Regression Derivation **26**
K.1 Proof of Proposition 1 26
K.2 Relation to other Online Logistic Regression approaches 28
K.3 Toy Experiment 29

L Architectural Details **30**

M Fast Online Deep Learning - Additional Details **31**
M.1 Proof of Proposition 2 31
M.2 Complexity analysis 33

A Broader Impact Statement

Our paper introduces a new Online Learning technique and improves existing training methodologies for online learning of deep neural networks. We show that co-training multiple learners can lead to significantly faster convergence as well as improved overall model performance at inference time.

A strongly positive outcome stemming from our contributions is the significant reduction of training time from quadratic complexity in network parameters down to linear complexity. This has profound consequences for training deep models online as it significantly decreases the necessary training compute. Thus the energy spent for training deep online learners is massively reduced.

Furthermore, the improved convergence speed and overall performance of our proposed technique MODL means that our model is more data efficient than existing models, leading to a moderately reduced need for large dataset sizes. Efficiently learning in one pass means that data does not need to be stored, which can protect the privacy rights of individuals and organizations. Training online without storing data can help companies comply with data protection laws and allow the consumer to have greater confidence that their fundamental human right of privacy is protected.

We are confident that our research effort offers more benefits in energy saving and data privacy as opposed to the risks posed by the usage of deep models. Furthermore, we point out that the risks posed by a potential deployment of our model can be hedged against due to the interpretability of some of our constituent learners. In our case, the potential to interpret model outputs is particularly strong for the weaker learners such as logistic regression.

B Additional Background

B.1 Model weighing approaches

Suppose we have a set of trained models. It is frequently suboptimal to discard all models except the best performing one. Approaches such as Bayesian model averaging (BMA) estimate the predicted quantity under each candidate model and then produce a weighted average estimate over all models. In BMA, the weights are determined according to the probability that each model represents the true data generating mechanism (DGM) given the data (46). However, BMA assumes that at least one element of the model set we are averaging over contains a correctly specified model that can truly encompass the DGM (44). This assumption is not true in general deep learning settings; this often leads to suboptimal generalization of BMA (34). We therefore search for alternative methods for building fundamental model architectures capable of combining multiple learners. One such approach is model stacking (47; 22). While stacking was originally only capable of producing point estimates, renewed interest has recently framed it from a Bayesian perspective (14; 33). Recent approaches cast selection of stacking weights as a Bayesian decision problem (50).

B.2 Online Learning with Missing Features

In this section we describe in detail the standard base architecture that underpins most modern online deep learning techniques. Algorithm 2 outlines the current online deep learning state-of-the-art base architecture, ODL, developed by Sahoo et al. (40) and refined in Aux-Drop (ODL) (3). This section reviews this architecture, which is based on a joint bi-level optimization objective. The learning (or selection) of the architecture is accomplished by attaching early exit classifiers to each hidden layer, and taking a weighted convex sum of all classifiers to produce the overall output. The weights attached to each classifier's output in the sum are defined as α and the network architecture is selected by optimizing these α values. Each alpha connects one of the hidden layers with the overall neural network output. The other optimization level handles the optimization of model parameters Θ. The algorithm oscillates between taking an optimization step in the network weights and a "hedge" (21) optimization step that effectively updates the architecture by assigning weights to the skip connections (see last step of Algorithm 2).

Consider a deep neural network with L hidden layers where each layer is connected to an early exit predictor $\mathbf{f}^{(i)}$. The prediction function $\mathbf{F}$ for the deep neural network is given by:

$$\mathbf{F}(\mathbf{x}) = \sum_{l=0}^{L} \alpha^{(l)} \mathbf{f}^{(l)}, \quad \text{where } \mathbf{f}^{(l)} = \text{softmax}(\mathbf{h}^{(l)} \boldsymbol{\Theta}^{(\boldsymbol{l})}), \tag{15}$$
$$\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)}),\ \forall l = 1, \ldots, L; \mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^{d_{\text{in}}}.$$

Here, $\boldsymbol{\Theta}^{(\boldsymbol{l})}$ is the parameter matrix of the early exit classifier $\mathbf{f}^{(l)}$, and $\mathbf{W}^{(l-1)}$ is the parameter matrix of the hidden layer that yields intermediate representation $\mathbf{h}^{(l)}$. Learning the parameters $\boldsymbol{\Theta}^{(\boldsymbol{l})}$ for classifiers $\mathbf{f}^{(l)}$ can be done via online gradient descent (OGD), where the input to the l^{th} classifier is $\mathbf{h}^{(l)}$. This is essentially standard backpropagation with learning rate η:

$$\boldsymbol{\Theta}_{t+1}^{(l)} \leftarrow \boldsymbol{\Theta}_{t}^{(l)} - \eta \nabla_{\boldsymbol{\Theta}_{t}^{(l)}} \mathcal{L}(\mathbf{F}(\mathbf{x}_t, y_t)). \tag{16}$$

Updating the feature representation parameters $\mathbf{W}^{(l)}$ potentially requires backpropagating through all classifiers $\mathbf{f}^{(l)}$. Thus, using the adaptive loss 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, the update rule for $\mathbf{W}^{(l)}$ is given by:

$$\mathbf{W}_{t+1}^{(l)} \leftarrow \mathbf{W}_{t}^{(l)} - \eta \sum_{j=l}^{L} \alpha^{(j)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t), \tag{17}$$

where $\nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t)$ is computed via backpropagation from error derivatives of $\mathbf{f}^{(j)}$. The summation in the gradient term starts at $j = l$ because the shallower classifiers do not depend on $\mathbf{W}^{(l)}$ for making predictions. This training technique is summarized in Algorithm 2.

When eq. (17) is implemented directly, a single backpropagation pass for this architecture incurs an $O(L^2)$ cost, which is evident from its implementation code[2]. We also demonstrate this in the proof of Proposition 2. This can grow prohibitively expensive for deep networks and has had major implications in the efficiency of subsequent research methods and applications that rely on ODL. For example, works as recent as Aux-Drop (3) build on this framework. In the next section, we first show how an equivalent update to eq. (17) can be derived by "rewiring" the network to reduce time complexity from $O(L^2)$ to $O(L)$ per backpropagation update. **This improved implementation of the base ODL architecture constitutes a secondary yet notable contribution of our work that we did not explore in the main text due to space limitations.** We carefully derive and prove the improved complexity of our implementation in the subsequent section.

C Fast Online Deep Learning

Fast ODL (FODL). During training, rather than backpropagating through L individual early exit classifier losses, we propose aggregating the outputs and backpropagating through a linear combination to obtain $\mathcal{L}_{\text{TOT}}$. Detailed architecture schematics and analysis of this are provided in Appendix M.1. This change in the network topology has important ramifications for training via backpropagation. In particular, this alters eq. (17) as follows:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \nabla_{W^{(l)}} \mathcal{L}_{\text{TOT}}, \quad \text{where} \tag{18}$$

$$\mathcal{L}_{\text{TOT}} = \left[\alpha^{(1)}, \ldots, \alpha^{(L)}\right] \left[\mathcal{L}(\mathbf{f}^{(1)}, y_t), \ldots, \mathcal{L}(\mathbf{f}^{(L)}, y_t)\right]^T = \sum_{j=1}^{L} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t)\,. \tag{19}$$

Proposition 2. *The updates described by eq. (17) and eq. (18) produce the same gradient update but the proposed training scheme has lower complexity. For a dataset of size n and a network with L layers, eq. (17) has training time complexity (due to backpropagation) $O(nL^2)$, but eq. (18) only requires $O(nL)$ computation.*

Proof. The proof is in Appendix M.1. □

[2]The most popular ODL implementation has 171 stars and 44 forks at the time of writing and has quadratic training complexity: https://github.com/alison-carrera/onn

D Algorithms

Algorithm 2 Online Deep Learning (ODL) using Hedge Backpropagation

Require: Learning rate Parameter: η, discount parameter β
Initialize: $F(\mathbf{x})$ with L hidden layers and $L+1$ classifiers $f^{(l)}$; $\alpha^{(l)} = \frac{1}{L+1}, \forall l = 0, \ldots, L$

for t = 1,...,T **do**
Receive instance $\mathbf{x}_t$ and predict $\hat{y}_t = F_t(\mathbf{x}_t) = \sum_{l=0}^{L} \alpha_t^{(l)} f_t^{(l)}$ via eq. (15)
Reveal true value y_t and calculate $\mathcal{L}_t^{(l)} = \mathcal{L}(\mathbf{f}_t^{(l)}(\mathbf{x}_t), y_t), \forall l, \ldots, L$;
Update $\Theta_{t+1}^{(l)}, \forall l = 0, \ldots, L$ via eq. (16) and $W_{t+1}^{(l)}, \forall l = 1, \ldots, L$ via eq. (17);
Update $\alpha_{t+1}^{(l)} = \alpha_t^{(l)} \beta^{\mathcal{L}_t^{(l)}}, \forall l = 0, \ldots, L$ and normalize $\alpha_{t+1}^{(l)}$ to sum to 1.
end for

Algorithm 3 Online Bayesian Logistic Regression

Require: Dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
Initialize: $p(\theta) = \mathcal{N}(\theta; \mathbf{m}_0, \mathbf{P}_0)$, $\mathbf{m}_0 = 0$, where $\mathbf{P_0} = \mathbf{I}$
Return $p(\theta|\{(\mathbf{x}_i, y_i)\}_{i=1}^n) = \mathcal{N}(\theta; \mathbf{m}_n, \mathbf{P}_n)$
for t = 1,...,n **do**
Receive instance: $\mathbf{x}_t$ and predict: $\hat{y}_t = \sigma(\mathbf{x}_t \mathbf{m}_{t-1})$
Reveal true value: y_t and update parameters:

$$\Xi_t = \mathbf{x}_t \mathbf{P}_{t-1} \mathbf{x}_t^\top + \mathbf{P} \left[(1 - \sigma(\mathbf{x}_t \mathbf{m}_{t-1}))\, \sigma\left(\mathbf{x}_t \mathbf{m}_{t-1}\right) \right]^2, \tag{20}$$

$$\mathbf{K}_t = \mathbf{P}_{t-1} \mathbf{X}_t^\top \left(1 - \sigma(\mathbf{x}_t \mathbf{m}_{t-1})\right) \sigma\left(\mathbf{x}_t \mathbf{m}_{t-1}\right) \Xi_t^{-1}, \tag{21}$$

$$\mathbf{m}_t = \mathbf{m}_{t-1} + \mathbf{K}_t [\mathbf{y}_t - \sigma\left(\mathbf{x}_t \mathbf{m}_{t-1}\right)], \tag{22}$$

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \mathbf{K}_t \Xi_t \mathbf{K}_t^\top. \tag{23}$$

end for

E Additional Experiments

In this section we provide additional experiments to verify four research claims empirically.

1. MODL outperforms Aux-Drop (ODL) for all feature missingness settings on large benchmarks (HIGGS, SUSY).
2. Removing any model component hurts model performance for all missingness levels on large benchmarks. This validates the choice of using multiple learners across the bias-variance trade-off spectrum.
3. Our improved implementation of ODL and Aux-Drop ODL, which we call Fast ODL, and abbreviate as FODL and Aux-Drop FODL, has lower time complexity without sacrificing any performance compared to the standard ODL and Aux-Drop (ODL) implementation.
4. Our proposed approach MODL performs better for a wider range of learning rates. This sensitivity experiment is key as learning rate selection is one of the most difficult hyperparameters to tune in Online Learning.

E.1 Feature Missingness Experiment

In this section we provide additional experiments in support of the summary results presented in the main text. More concretely, we provide a full feature missningess study with 13 unique $p_{\mathbf{f}}$ values for the large benchmarks HIGGS and SUSY. As shown in Tab. 8, our model convincingly outperforms for all settings. We observe that the advantage of our method increases as more features become available.

Table 8: Comparison on HIGGS and SUSY for various feature probabilities p_{f}. The metric is the mean ($\pm$ standard deviation) cumulative error in thousands (5 runs).

$p_{\mathbf{f}}$	**HIGGS**	
	Aux-Drop (ODL)	MODL (ours)
.01	440.2 ± 0.1	**439.6 ± 0.1**
.05	440.0 ± 0.1	**439.5 ± 0.2**
.10	440.0 ± 0.2	**438.5 ± 0.1**
.20	438.4 ± 0.1	**435.6 ± 0.4**
.30	435.1 ± 0.2	**432.3 ± 0.3**
.40	432.0 ± 0.3	**428.4 ± 0.3**
.50	427.4 ± 0.7	**422.8 ± 0.4**
.60	423.2 ± 0.5	**422.8 ± 0.2**
.70	418.5 ± 0.7	**409.5 ± 0.4**
.80	411.8 ± 0.4	**399.6 ± 0.3**
.90	405.6 ± 0.7	**387.0 ± 0.3**
.95	399.4 ± 1.0	**377.2 ± 0.5**
.99	392.1 ± 1.0	**366.5 ± 0.6**

$p_{\mathbf{f}}$	**SUSY**	
	Aux-Drop (ODL)	MODL (ours)
.01	285.0 ± 0.1	**283.0 ± 0.1**
.05	283.3 ± 0.2	**281.0 ± 0.1**
.10	280.6 ± 0.5	**278.0 ± 0.1**
.20	274.9 ± 0.9	**271.9 ± 0.1**
.30	269.0 ± 0.7	**265.5 ± 0.1**
.40	262.8 ± 0.9	**258.9 ± 0.1**
.50	256.7 ± 1.0	**252.0 ± 0.1**
.60	250.0 ± 0.9	**244.7 ± 0.2**
.70	243.9 ± 0.7	**238.0 ± 0.2**
.80	237.0 ± 0.7	**230.6 ± 0.1**
.90	230.0 ± 0.7	**222.3 ± 0.2**
.95	226.2 ± 0.4	**217.5 ± 0.1**
.99	222.2 ± 0.2	**212.2 ± 0.2**

E.2 Model Ablation Study

In this section we empirically validate our proposed model MODL. As shown in Table 9 the results in the main paper persist in the additional feature missningness settings, further validating our conclusion that all of our model components are necessary.

Table 9: Ablation of proposed model components. We show that for each learner that we add the overall model performance improves. OLR refers to online logistic regression, "Set" is short for Set Learner model.

$p_{\mathbf{f}}$	HIGGS			SUSY		
	OLR + MLP	OLR + Set	MODL (ours)	OLR + MLP	OLR + Set	MODL (ours)
.01	442.8±0.2	450.2±0.5	**439.6 ± 0.1**	285.3±0.1	332.6±2.1	**283.0 ± 0.1**
.05	442.7±0.1	449.9±0.4	**439.5 ± 0.2**	283.3±0.1	334.7±1.9	**281.1 ± 0.1**
.10	441.9±0.1	450.1±0.3	**438.5 ± 0.1**	280.0±0.1	337.8±1.4	**278.0 ± 0.1**
.20	439.9±0.1	450.3±0.2	**435.6 ± 0.4**	274.0±0.1	352.4±0.3	**271.9 ± 0.1**
.30	437.4±0.1	450.1±0.3	**432.3 ± 0.3**	267.9±0.1	355.4±0.3	**265.5 ± 0.1**
.40	434.9±332	450.3±404	**428.4 ± 302**	261.7±0.1	338.9±0.4	**258.9 ± 0.1**
.50	430.3±193	450.1±93	**422.8 ± 386**	255.0±0.1	343.3±0.7	**252.0 ± 0.1**
.60	425.4±178	449.9±137	**422.8 ± 244**	248.3±0.1	336.6±0.5	**244.7 ± 0.2**
.70	420.5±484	451.0±122	**409.5 ± 424**	241.3±0.1	337.9±0.4	**238.0 ± 0.1**
.80	411.8±106	449.6±219	**399.6 ± 256**	234.3±0.1	326.3±0.4	**230.6 ± 0.1**
.90	401.4±517	449.8±58	**387.0 ± 348**	226.5±0.2	322.0±0.4	**222.3 ± 0.2**
.95	394.1±565	448.5±175	**377.2 ± 537**	222.1±0.1	320.9±0.4	**217.5 ± 0.1**
.99	386.7±133	447.1±163	**366.4 ± 588**	217.4±0.1	316.4±0.7	**212.2 ± 0.2**

E.3 Training Time Additional Settings

In this section, we explore the wall clock training time reduction offered by our proposed fast training scheme on large datasets. We note that the training time savings range from 30% to over 80%.

Table 10: Time comparison between Aux-DropODL and FastAux-DropODL on HIGGS and SUSY for $p = 0.5$ (various layer and embedding dimensions).

Experiment	**Aux-Drop (ODL)**	**Aux-Drop (FODL)**	**Time ODL vs. FODL**
HIGGS ($L = 5, E = 25$)	431.9 ± 0.5	431.4 ± 0.5	5:35:10 vs. 3:34:55
HIGGS ($L = 5, E = 50$)	429.6 ± 0.5	429.6 ± 0.6	5:41:05 vs. 3:38:18
HIGGS ($L = 5, E = 100$)	428.0 ± 0.3	427.9 ± 0.3	5:45:56 vs. 3:38:42
HIGGS ($L = 11, E = 25$)	429.8 ± 0.7	429.7 ± 0.4	22:44:06 vs. 6:21:58
HIGGS ($L = 11, E = 50$)	427.4 ± 0.5	427.4 ± 0.7	16:04:59 vs. 4:29:05
HIGGS ($L = 11, E = 100$)	426.9 ± 0.3	426.6 ± 0.3	20:05:39 vs. 6:32:20
HIGGS ($L = 20, E = 25$)	431.4 ± 0.5	431.0 ± 0.5	57:04:12 vs. 9:23:13
HIGGS ($L = 20, E = 50$)	429.2 ± 0.4	429.4 ± 0.4	54:04:10 vs. 8:43:11
HIGGS ($L = 20, E = 100$)	427.6 ± 0.5	427.8 ± 0.4	63:19:05 vs. 8:58:51
SUSY ($L = 5, E = 25$)	257.3 ± 1.0	257.3 ± 1.2	5:33:53 vs. 2:59:32
SUSY ($L = 5, E = 50$)	256.6 ± 0.9	256.6 ± 0.9	5:37:42 vs. 3:03:09
SUSY ($L = 5, E = 100$)	255.9 ± 0.4	255.8 ± 0.5	5:07:21 vs. 3:41:03
SUSY ($L = 11, E = 25$)	257.2 ± 0.9	257.5 ± 1.1	22:41:05 vs. 6:19:03
SUSY ($L = 11, E = 50$)	257.0 ± 1.2	256.7 ± 1.0	23:04:11 vs. 6:54:21
SUSY ($L = 11, E = 100$)	255.9 ± 0.8	256.1 ± 0.8	20:05:06 vs. 6:29:58
SUSY ($L = 20, E = 25$)	258.6 ± 1.1	258.5 ± 1.3	36:39:13 vs. 8:44:24
SUSY ($L = 20, E = 50$)	257.4 ± 0.9	257.4 ± 0.9	39:59:38 vs. 8:01:53
SUSY ($L = 20, E = 100$)	257.3 ± 1.0	257.1 ± 1.0	43:07:45 vs. 8:19:44

E.4 Learning Rate Sensitivity Experiments

In this section we demonstrate that our model has stable and better performance than the baseline approaches for a broad range of learning rates. This is due to the incorporation of learners without a backpropagation update. This shields the overall architecture from poorer learning rate selections. Note that if we set the learning rate too high the other learners may become unstable. Protection from bad learning rates thus has limits (i.e., cannot train with very fast learning rates such as 0.5). The conclusion from this section and Tab. 11 is that for any reasonable learning rate selection for deep models our model outperforms.

Table 11: Error in HIGGS and SUSY for various learning rates with fixed probability of unreliable features $p_f = 0.99$. The metric is reported as the mean ± standard deviation of the number of errors in 5 runs (11 layer Aux networks).

lr	**HIGGS**		**SUSY**	
	Aux-Drop (ODL)	MODL (ours)	Aux-Drop (ODL)	MODL (ours)
0.00005	470.9±0.8	**426.1± 1.2**	391.9 ± 67	**225.5 ± 0.7**
0.0001	470.1±0.7	**415.4±0.8**	333.4±62	**222.2 ± 0.4**
0.0005	442.8±3.5	**392.7±0.6**	278.6±21	**216.6 ± 0.2**
0.001	402.1±6.4	**366.5±0.6**	225.9±1.9	**212.2 ± 0.2**
0.005	439.4±11.4	**383.3±0.9**	253.6±11	**215.0 ± 0.3**
0.01	389.9±1.8	**368.9±0.5**	222.7±0.9	**212.8 ± 0.1**
0.05	**392.1±0.8**	421.1±0.5	**222.4±0.2**	227.2 ± 0.1
0.1	**418.7±5.9**	461.9±0.3	**227.6±0.5**	242.4 ± 0.4

F Dataset Statistics

Our dataset pre-processing for *german*, *svmguide3*, *magic04*, *a8a*, *SUSY* and *HIGGS* follows exactly the steps from Agarwal et al. (3). The data is publicly available for download online[3]. For CIFAR-10 we use the standard training set that consists of 50,000 images. Image pixel intensities are normalized to fall within $[0, 1]$ range. For I-MNIST there used to exist an online repository of the dataset. However, this repository appears to have been removed. As a result we rebuild the dataset using the source code that originally generated the data from L. Bottou[4]. We use the default configuration settings provided in the repository. We provide summary dataset statistics in Tab. 12. The use of diverse dataset sizes aims to illustrate the strength of our approach in settings with low data (one thousand), moderate amount (tens of thousands) and large data (millions).

Table 12: Dataset statistics for the data used in our experiments.

Dataset	Size	Feature Size	Task
german	1000	24	Classification
svmguide3	1243	21	Classification
magic04	19020	10	Classification
a8a	32561	123	Classification
CIFAR-10	50000	32×32×3	Classification
I-MNIST	1000000	28×28	Classification
SUSY	1000000	8	Classification
HIGGS	1000000	21	Classification

G Score Sum Validation Implementation Details

Due to space constraints in the main paper we limited discussion of the details of the baseline designs for this ablation study. Here we add concrete details for each baseline. The results for the baselines described below, comparing them against our method, appear in Table 4 of the original paper.

Denote each learner output as f_i. Note that for this ablation study we fix the learner pool and the architectures of the learner models to be the same as MODL in Tables 1, 2. Below, we provide the detailed description of each baseline.

Mixture of experts (MoE) model learns a gating mechanism to combine model scores f_i. Note that standard MoE practice is to combine logit probabilities rather than latent scores so we apply a softmax function to each learner to produce a valid probability distribution. The gating network G, parameterized by Θ is a 2 layer softmax network, yielding the output $G(x; \Theta)$:

$$F_{\text{MoE}}(x; \Theta, \{W_i\}_{i=1}^{N}) = \sum_{i=1}^{N} G(x; \Theta)_i [\text{softmax}(f_i)], \tag{24}$$

$$G(x; \Theta)_i = \text{softmax}(g(x; \Theta))_i = \frac{\exp(g(x; \Theta)_i)}{\sum_{j=1}^{N} \exp(g(x; \Theta)_j)}, \tag{25}$$

where $g(\cdot)$ is the value at the penultimate layer of the neural network parameterizing $G(x, \Theta)$. Note that this approach is a generic Dense MoE that has been commonly applied in the literature ("Mixture of experts: a literature survey", Masoudnia and Ebrahimpour, 2014). Note that backpropagation steps are taken from the output of F_{MoE}, and therefore the models are trained jointly here.

Ensemble (Deep Ensembles - Lakshminarayanan et al. (32)) model combines the predicted class probabilities from each model as follows:

[3]Available here: https://github.com/Rohit102497/Aux-Drop/tree/main/Code/Datasets
[4]Available here: https://leon.bottou.org/projects/infimnist

$$F_{\text{DE}}(x) = \frac{1}{N}\sum_{i=1}^{N}\text{softmax}(f_i), \tag{26}$$

where the models are trained independently (backpropagation passes are separate).

Greedily Weighing is the online weighted ensemble method that weighs models proportionally to their online accuracy within a sliding window. Concretely, this takes the form:

$$F_{\text{Greedy}}(x) = \sum_{i=1}^{N}(\text{softmax}([\lambda_t])_i)\ \text{softmax}(f_i), \tag{27}$$

$$[\lambda_t]_i = \sum_{j=t-K}^{t-1}\frac{\mathbb{I}[f_i(x_j) = y_j]}{K}, \tag{28}$$

where λ_t is a vector of length N with the running accuracy of each of the model within the sliding window of size K, $[\lambda_t]_i$ is the i-th index of the vector (and contains the running accuracy of i-th model) and $\mathbb{I}[\cdot]$ is an indicator function. In our experiments we chose $K = 100$ as a reasonable tradeoff to achieve responsiveness to model performance change but also robustness to temporary fluctuations in model performance as the models converge. If K is set too low, e.g., below 10, then the method can become unstable. Again, the ensemble methods here are trained independently (backpropagation passes are separate).

Multiplication baseline interprets each model output (after it is projected to a valid probability distribution) as an independent estimate of the class probability. Hence, in order to obtain a distribution over all models, it multiplies the constituent distributions and re-normalizes. For numerical stability this is done via logarithm addition, which also has a clear interpretaion as the sum ensemble in the logit domain:

$$F_{\text{Mult}}(x) = \text{softmax}\left(\sum_{i=1}^{N}\log\text{softmax}(f_i)\right). \tag{29}$$

Here, models are entangled via the softmax, and the backpropagation pass operates on the combined output and thus trains the models jointly.

H Hyperparameters and Computational Resources

H.1 Hyperparameters

Aux-Drop (ODL): We used the official repository of Aux-Drop (ODL)[5] with the tuned hyperparameters from the original paper by Agarwal et al. (3) without making any changes. Namely, for the small datasets in our experiments we run 6-layer networks with learning rates 0.1 for german and svmguide3 and 0.01 for magic04 and a8a. For the large datasets, we used 11 layer networks with learning rate equal to 0.05. For image datasets we run Aux-Drop using 20 learning rates between $[10e-6, 10e-2]$ and reported the results of the best setup $(5e-4)$. For all datasets we put the AuxLayer in the third layer of the network. For *german*, *svmguide3*, *magic04*, *SUSY* and *HIGGS* the capacity of the hidden layer is 50 and in the capacity of AuxLayer is 100. For *a8a* the AuxLayer has 400 units. For CIFAR-10 and I-MNIST we use multiple hidden unit layer capacities $\{100, 250, 500\}$ and report the results on the best one (250).

For all experiments we used the recommended layer size and AuxLayer dimension and position in the network. Additionally, the ODL discount rate was set to $\beta = 0.99$ and the smoothing rate was set to $s = 0.2$.

MODL: We used the exact same architecture in all experiments. Specifically, the online logistic regression learner is implemented identically for all datasets and the MLP has 3 hidden layers with 250 neurons. The set

[5]Available here: https://github.com/Rohit102497/Aux-Drop

learner consists of 6 blocks with 3 layers per block. The width of each layer is set to 128 neurons. The overall performance is stable for various layer widths. For the MLP and the set learner we observe that among the capacities of $\{100, 250, 500\}$ neurons per layer the best performance is given by 250 units but the difference is not very large. For the set learner we validated that the number of blocks in the encoder and decoder (candidate values $\{2, 3, 4, 5, 6, 7, 8\}$) and the number of layers per block (candidate values $\{2, 3, 4, 5\}$) do not have a huge performance impact, though deeper architectures perform better for larger datasets and more narrow ones have an edge in smaller datasets. Our selected setup for all datasets has good all around performance.

Table 13: Hyperparameters for MODL in all experiments.

Dataset	Learning Rate	Layers MLP	MLP Layer Width	Blocks Set Learner	Layers Block	Block Layer Width
german	0.01	3	250	6	3	250
svmguide3	0.01	3	250	6	3	250
magic04	0.001	3	250	6	3	250
a8a	0.001	3	250	6	3	250
CIFAR-10	0.00005	3	250	6	3	250
I-MNIST	0.00005	3	250	6	3	250
SUSY	0.005	3	250	6	3	250
HIGGS	0.005	3	250	6	3	250

H.2 Computational Resources

We run our experiments on a HP DL-580 Gen 7 server equipped with Intel Xeon E7-4870 2.40GHz 10-Core 30MB LGA1567 CPUs. In total we used 160 CPUs for our experiments. Note that using GPUs in the online learning setting is not effective as the batch size is 1. Therefore, given that our task is purely sequential we opt for a CPU server that is actually faster than a GPU server given the nature of the online learning task.

I Bayesian Filters for Online Regression

In this section we present the key theoretical result that underpins our online logistic regression learner. This result stems from De Finetti's representation theorem. This key result allows us to derive recursive algorithms that update prior-posterior parameters without storing observations, an essential element in any dataset-memoryless online algorithm. We then review a standard result for online linear regression which we then generalize via linearization and a Normal approximation to obtain the online logistic regression learner we use in our algorithm.

I.1 De Finetti Representation Theorem

For $n \geq 1$ the de Finetti representation for the joint mass or density of exchangeable discrete random variables $Y_1, \ldots, Y_n$ is given by

$$p_{Y_1,\ldots,Y_n}(y_1, \ldots, y_n) = \int_\Theta \prod_{i=1}^{n} f_Y(y_i; \theta)\pi_0(d\theta)$$

where $p_Y(y; \theta)$ is a mass function in y, and θ is a parameter lying in a space $\Theta \in \mathbb{R}^p$, for some (prior) distribution $\pi_0(d\theta)$ defined on θ, where we may interpret Θ as the smallest set such that $\int_\Theta \pi_0(d\theta) = 1$. Using basic properties of probability we may derive an expression for the posterior predictive distribution for Y_{n+1} (another element of the infinitely exchangeable sequence) conditional on $Y_1 = y_1, \ldots, Y_n = y_n$. This defines the posterior density, $\pi_n(d\theta)$, as an updated version of $\pi_0(d\theta)$. We may re-write the posterior predictive distribution using the definition of conditional probability as

$$p_n(y_{n+1}) = \frac{p_0(y_1, \cdots, y_n, y_{n+1})}{p_0(y_1, \cdots, y_n)} = \int f_Y(y_{n+1}; \theta)\pi_n(\theta)\, d\theta \tag{30}$$

where

$$\pi_n(\theta) = \frac{1}{p_0(y_1, \cdots, y_n)} \prod_{i=1}^{n} f(y_i; \theta)\pi_0(\theta), \tag{31}$$

is the posterior. Note that this is the same form as the De Finetti representation with an updated version of π_0.

This result is important as it shows that the posterior predictive is the same whether we consider the initial prior and the likelihood of a full batch on n observations or if we use an iterative update rule to modify the prior at each time step. Thus, we may simply update the prior for the n+1-th observation to be the posterior after the n-th observation without the need to maintain any of the previous observations in memory. This allows us to derive recursive algorithms that update prior-posterior parameters without storing observations. Essentially, as long as we maintain a statistic that is sufficient in the Bayesian sense, the posterior distribution depends on the data only through the observed value of the statistic and the data may be discarded.

I.2 Bayesian Linear Regression

For a dataset $\mathcal{D} = \{(t_1, y_1), \ldots, (y_N, t_N)\}$, consider a classic regression model of the form:

$$y_k = \mathbf{x}_k \theta + \varepsilon_k, \tag{32}$$

where y_k is a scalar outcome, $\theta \in \mathbb{R}^{d\times 1}$ is the parameter of interest, $\mathbf{x}_k \in \mathbb{R}^{1\times d}$ is the feature vector and $\varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ is normally distributed noise. To simplify the notation we may stack all feature vectors in the designer matrix $\mathbf{X} \in \mathbb{R}^{N\times d}$ and the corresponding outcomes in a vector $\mathbf{y}_k \in \mathbb{R}^N$.

Though the problem can be solved in a purely least squares fashion or equivalently via a maximum-likelihood approach, we employ a standard Bayesian formulation. Assigning the prior,

$$\pi_0(\theta) = \mathcal{N}(\theta \,|\, \mathbf{m}_0, \mathbf{P}_0)\,, \tag{33}$$

for some $\mathbf{m}_0 \in \mathbb{R}^d, \mathbf{P}_0 \in \mathbb{R}^{d\times d}$. and modeling $p(\mathbf{y}_k \,|\, \theta)$ as

$$p(y_k \,|\, \theta) = \mathcal{N}(y_k \,|\, \mathbf{x}_k\theta, \sigma^2), \tag{34}$$

we may then apply Bayes' rule to recover the posterior distribution over the parameter:

$$p(\theta \,|\, y_{1:N}) \propto p(\theta) \prod_{k=1}^{N} p(y_k \,|\, \theta) = \mathcal{N}(\theta \,|\, \mathbf{m}_0, \mathbf{P}_0) \prod_{k=1}^{N} \mathcal{N}(y_k \,|\, \mathbf{x}_k\theta, \sigma^2) \tag{35}$$

$$= \mathcal{N}(\theta \,|\, \mathbf{m}_N, \mathbf{P}_N)\,. \tag{36}$$

The last equality follows from the well known result that normal priors and normal likelihoods yield normal posteriors. In fact, it can be shown that

$$\mathbf{m}_N = \left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X}\right]^{-1}\left[\frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{y} + \mathbf{P}_0^{-1}\mathbf{m}_0\right], \tag{37}$$

$$\mathbf{P}_N = \left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X}\right]^{-1}. \tag{38}$$

I.3 Online Bayesian Linear Regression

A disadvantage of full batch regression is that it requires keeping a large matrix $\left[\mathbf{P}_0^{-1} + \frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X}\right]^{-1}$ in memory and inverting it. To address these concerns, we can use an efficient and recursive version of the algorithm. From the recursive de Finetti represention result we presented earlier, we may alternatively calculate a posterior at step n-1 and treat it as a prior for step n.

$$p(\theta \,|\, y_{1:n}) \propto p(\theta \,|\, y_{1:n-1})p(y_n \,|\, \theta) = \mathcal{N}(\theta \,|\, \mathbf{m}_{n-1}, \mathbf{P}_{n-1})\mathcal{N}(y_k \,|\, \mathbf{x}_k\theta, \sigma^2) \tag{39}$$

$$= \mathcal{N}(\theta \,|\, \mathbf{m}_n, \mathbf{P}_n), \tag{40}$$

where

$$\mathbf{m}_n = \left[\mathbf{P}_{n-1}^{-1} + \frac{1}{\sigma^2}\mathbf{X}_n^\top\mathbf{X}_n\right]^{-1}\left[\frac{1}{\sigma^2}\mathbf{X}_n^\top\mathbf{y}_n + \mathbf{P}_{n-1}^{-1}\mathbf{m}_{n-1}\right], \tag{41}$$

$$\mathbf{P}_n = \left[\mathbf{P}_{n-1}^{-1} + \frac{1}{\sigma^2}\mathbf{X}_n^\top\mathbf{X}_n\right]^{-1}. \tag{42}$$

We use the notation, $\mathbf{X}_n$ and $\mathbf{y}_n$ to indicate the n-th rows of $\mathbf{X}$, $\mathbf{y}$, respectively.

The matrix inversion lemma[6], also known as the Woodbury matrix identity, states that:

$$\mathbf{P}_n = \left[\mathbf{P}_{n-1}^{-1} + \frac{1}{\sigma^2}\mathbf{X}_n^\top \mathbf{X}_n\right]^{-1} = \mathbf{P}_{n-1} - \mathbf{P}_{n-1}\mathbf{X}_n^\top \left[\mathbf{X}_n \mathbf{P}_{n-1}\mathbf{X}_n^\top + \sigma^2\right]^{-1} \mathbf{X}_n \mathbf{P}_{n-1}. \tag{43}$$

Note that the inverse term in the expression on the right hand side is a scalar and thus easy to compute.

The recursive set of equations takes the form:

$$S_n = \mathbf{X}_n \mathbf{P}_{n-1}\mathbf{X}_n^\top + \sigma^2, \tag{44}$$
$$\mathbf{K}_n = \mathbf{P}_{n-1}\mathbf{X}_n^\top S_n^{-1}, \tag{45}$$
$$\mathbf{m}_n = \mathbf{m}_{n-1} + \mathbf{K}_n[\mathbf{y}_n - \mathbf{X}_n \mathbf{m}_{n-1}], \tag{46}$$
$$\mathbf{P}_n = \mathbf{P}_{n-1} - \mathbf{K}_n S_n \mathbf{K}_n^\top. \tag{47}$$

J Logistic Regression

Consider a Generalized Linear Model (GLM) with link function $g(\cdot) = \text{logit}^{-1}(\cdot)$. For binary classification we define $p(y \,|\, x) = \mathbb{E}_{Y|X}[y \,|\, \mathbf{x}] = g^{-1}(\mathbf{x}\theta) = \sigma(\mathbf{x}\theta) = \mu$, $\sigma(\cdot)$ is the sigmoid function. The problem of finding the optimal parameter can be solved by maximizing the (log) likelihood numerically as it is convex. More precisely the derivative of the log-likelihood and the Hessian can be shown to be

$$\dot{\ell}_n(\theta) = \mathbf{X}^\top(\mathbf{y} - \boldsymbol{\mu}), \tag{48}$$
$$\ddot{\ell}_n(\theta) = -\mathbf{X}^\top \mathbf{D}\mathbf{X}, \tag{49}$$

where $\boldsymbol{\mu} = [\sigma(\mathbf{X}_1\theta), \ldots, \sigma(\mathbf{X}_n\theta)]$ is a vector containing the model prediction for each input, $\mathbf{D}$ is a diagonal matrix with the diagonal entries $[\sigma(\mathbf{X}_1\theta)(1-\sigma(\mathbf{X}_1\theta)), \ldots, \sigma(\mathbf{X}_n\theta)(1-\sigma(\mathbf{X}_n\theta))]$. Since this is a convex problem, it has a unique solution that can be recovered via standard convex optimization techniques.

K Online Logistic Regression Derivation

K.1 Proof of Proposition 1

Consider an input $\mathbf{z} \in \mathbb{R}^n$ and output $\mathbf{y} \in \mathbb{R}^m$, that are related via an invertible function $h(\cdot) : \mathbb{R}^n \to \mathbb{R}^m$

$$\mathbf{z} \sim \mathcal{N}(\mathbf{z} \,|\, \mathbf{m}, \mathbf{P}) \tag{50}$$
$$\mathbf{y} = h(\mathbf{z}) + \varepsilon_t, \tag{51}$$

where we model the output as receiving additive noise $\varepsilon \sim \mathcal{N}(\varepsilon_t \,|\, 0, \boldsymbol{\Sigma}_t)$ that is independent of the input. By standard random variable transformation theory we know that $p(\mathbf{y}) = \mathcal{N}(h^{-1}(\mathbf{y}) \,|\, \mathbf{m}, \mathbf{P})|\mathcal{J}(\mathbf{y})|^{-1}$, where $|\mathcal{J}(\mathbf{y})|$ is the Jacobian of h evaluated at $\mathbf{y}$.

A challenge in analyzing this model is that it is non-linear. A standard technique to approximate $p(\mathbf{y})$ is to apply a first order Taylor approximation and linearize the function locally around the mean of the normal $\mathcal{N}(\mathbf{z} \,|\, \mathbf{m}, \mathbf{P})$ with some random perturbation $\delta\mathbf{z} \sim \mathcal{N}(\delta\mathbf{z} \,|\, 0, \mathbf{P})$:

$$h(\mathbf{z}) = g(\mathbf{m} + \delta\mathbf{z}) \approx g(\mathbf{m}) + \mathcal{J}(\mathbf{m})\delta\mathbf{z} = \widehat{h}(\mathbf{z}). \tag{52}$$

Firstly, note that $\mathbb{E}[h(\mathbf{z})] \approx \mathbb{E}[h(\mathbf{m})] + \mathcal{J}(\mathbf{m})\mathbb{E}[\delta\mathbf{z}] = \mathbb{E}[h(\mathbf{m})]$. Similarly, for the covariance matrix it follows that

[6] https://en.wikipedia.org/wiki/Woodbury_matrix_identity

$\text{cov}(h(\mathbf{x})) = \mathbb{E}\left[(h(\mathbf{z}) - \mathbb{E}[h(\mathbf{z})])(h(\mathbf{z}) - \mathbb{E}[h(\mathbf{z})])^\top\right] \approx \mathcal{J}(\mathbf{m})\mathbf{P}\mathcal{J}(\mathbf{m})^\top$. To show this in detail consider:

$$\begin{aligned}
\text{cov}(h(\mathbf{z})) &= \mathbb{E}\left[(h(\mathbf{z}) - \mathbb{E}[h(\mathbf{z})])(h(\mathbf{z}) - \mathbb{E}[h(\mathbf{z})])^\top\right] && (53)\\
&\approx \mathbb{E}\left[(h(\mathbf{z}) - h(\mathbf{m}))(h(\mathbf{z}) - h(\mathbf{m}))^\top\right] && (54)\\
&= \mathbb{E}\left[(h(\mathbf{m}) + \mathcal{J}(\mathbf{m})\delta\mathbf{z} - h(\mathbf{m}))(h(\mathbf{m}) + \mathcal{J}(\mathbf{m})\delta\mathbf{z} - h(\mathbf{m}))^\top\right] && (55)\\
&= \mathbb{E}\left[(\mathcal{J}(\mathbf{m})\delta\mathbf{z})(\mathcal{J}(\mathbf{m})\delta\mathbf{z})^\top\right] && (56)\\
&= \mathcal{J}(\mathbf{m})\mathbb{E}\left[(\delta\mathbf{z})(\delta\mathbf{z})^\top\right]\mathcal{J}(\mathbf{m})^\top && (57)\\
&= \mathcal{J}(\mathbf{m})\mathbf{P}\mathcal{J}(\mathbf{m})^\top. && (58)
\end{aligned}$$

Combining the previous results we obtain an approximate joint distribution of $\mathbf{z}, \mathbf{h}(\mathbf{z})$:

$$p\left(\begin{bmatrix}\mathbf{z}\\ \mathbf{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{m}\\ h(\mathbf{m})\end{bmatrix}, \begin{bmatrix}\mathbf{P} & [\mathbf{P}\mathcal{J}(\mathbf{m})]^\top\\ \mathbf{P}\mathcal{J}(\mathbf{m}) & \mathbf{\Sigma}_t + \mathcal{J}(\mathbf{m})\mathbf{P}\mathcal{J}^\top(\mathbf{m})\end{bmatrix}\right). \tag{59}$$

In the case of input features that are multiplied by the regression parameters, we have $h(\mathbf{z}) \stackrel{\text{def}}{=} h(\mathbf{x}\theta)$ where the input feature vector $\mathbf{x}$ is multiplied by the weights vector θ, *i.e.*, $\mathbf{z} \stackrel{\text{def}}{=} \mathbf{x}\theta$. This modifies the joint distribution we just derived:

$$\begin{aligned}
\theta &\sim \mathcal{N}(\theta \,|\, \mathbf{m}, \mathbf{P}) && (60)\\
\mathbf{y} &= h(\mathbf{x}\theta) + \varepsilon, && (61)
\end{aligned}$$

$$p\left(\begin{bmatrix}\theta\\ \mathbf{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{m}\\ h(\mathbf{m}\theta)\end{bmatrix}, \begin{bmatrix}\mathbf{P} & [\mathbf{P}\mathbf{x}\mathcal{J}(\mathbf{m})]^\top\\ \mathbf{P}\mathbf{x}\mathcal{J}(\mathbf{m}) & \mathbf{\Sigma}_t + \mathcal{J}(\mathbf{m})\mathbf{x}\mathbf{P}\mathbf{x}^\top\mathcal{J}^\top(\mathbf{m})\end{bmatrix}\right). \tag{62}$$

Then, from the joint distribution $p(\theta, \mathbf{y})$ we may obtain the conditional $p(\theta \,|\, \mathbf{y})$, which has a well known closed form for jointly normal random variables,

$$\begin{aligned}
p(\theta \,|\, \mathbf{y} = y_k) &= \mathcal{N}(\theta \,|(\mathbf{m}', \mathbf{P}'), \quad \text{where} && (63)\\
\mathbf{m}' &= \mathbf{m} + [\mathbf{P}\mathbf{x}\mathcal{J}(\mathbf{m})]^\top[\mathbf{\Sigma}_k + \mathcal{J}(\mathbf{m})\mathbf{x}\mathbf{P}\mathbf{x}^\top\mathcal{J}^\top(\mathbf{m})]^{-1}(\mathbf{y}_k - h(\mathbf{m}\theta)) && (64)\\
\mathbf{P}' &= \mathbf{P} - [\mathbf{P}\mathbf{x}\mathcal{J}(\mathbf{m})]^\top[\mathbf{\Sigma}_k + \mathcal{J}(\mathbf{m})\mathbf{x}\mathbf{P}\mathbf{x}^\top\mathcal{J}^\top(\mathbf{m})]^{-1}[\mathbf{P}\mathbf{x}\mathcal{J}(\mathbf{m})] && (65)
\end{aligned}$$

Thus we obtain a normal posterior over the weights. But observe that this is just another recursive normal prior — normal likelihood model. Thus we may obtain recursive form updates similar to those we derived for the linear regression earlier.

$$\begin{aligned}
p(\theta \,|\, y_{1:n}) &\propto p(\theta \,|\, y_{1:n-1})p(y_n \,|\, \theta) = \mathcal{N}(\theta \,|\, \mathbf{m}_{n-1}, \mathbf{P}_{n-1})\mathcal{N}(y_n \,|\, h(\mathbf{x}_n\theta), \sigma^2) && (66)\\
&= \mathcal{N}(\theta \,|\, \mathbf{m}_n, \mathbf{P}_n), && (67)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{m}_n &= \mathbf{m}_{n-1} + \frac{[\mathbf{P}_{n-1}\mathbf{x}_n^\top\mathcal{J}(\mathbf{m}_{n-1})]^\top[\mathbf{y}_n - h\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)]}{\mathbf{\Sigma}_n + \mathcal{J}(\mathbf{m}_{n-1})\mathbf{x}_n\mathbf{P}_{n-1}\mathbf{x}_n^\top\mathcal{J}^\top(\mathbf{m}_{n-1})}, && (68)\\
\mathbf{P}_n &= \mathbf{P}_{n-1} - \frac{\mathcal{J}^\top(\mathbf{m}_{n-1})\mathbf{P}_{n-1}\mathbf{x}_n^\top\mathbf{x}_n\mathbf{P}_{n-1}^\top\mathcal{J}(\mathbf{m}_{n-1})}{\mathbf{\Sigma}_n + \mathcal{J}(\mathbf{m}_{n-1})\mathbf{x}_n\mathbf{P}_{n-1}\mathbf{x}_n^\top\mathcal{J}^\top(\mathbf{m}_{n-1})}. && (69)
\end{aligned}$$

Online Binary Logistic Regression Concretely, for a binary logistic regression $h(\cdot) \stackrel{\text{def}}{=} \text{logit}^{-1}(\mathbf{x}\theta) = \sigma(\mathbf{x}\theta)$ we can calculate the Jacobian to obtain a closed form update.

$$p(\theta \,|\, y_{1:n}) = \mathcal{N}(\theta \,|\, \mathbf{m}_n, \mathbf{P}_n), \tag{70}$$

$$\begin{aligned}
\mathbf{m}_n &= \mathbf{m}_{n-1} + \frac{\mathbf{P}_{n-1}\left[(1 - \sigma(\mathbf{x}_n\mathbf{m}_{n-1}))\,\sigma\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)\right]\mathbf{x}_n^\top[\mathbf{y}_n - \sigma\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)]}{\mathbf{\Sigma}_n + \mathbf{P}_{n-1}\left[(1 - \sigma(\mathbf{x}_n\mathbf{m}_{n-1}))\,\sigma\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)\right]^2}, && (71)\\
\mathbf{P}_n &= \mathbf{P}_{n-1} - \frac{\mathbf{P}_{n-1}\mathbf{x}_n^\top\mathbf{x}_n\mathbf{P}_{n-1}^\top\left[(1 - \sigma(\mathbf{x}_n\mathbf{m}_{n-1}))\,\sigma\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)\right]^2}{\mathbf{\Sigma}_n + \mathbf{P}_{n-1}\left[(1 - \sigma(\mathbf{x}_n\mathbf{m}_{n-1}))\,\sigma\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)\right]^2}. && (72)
\end{aligned}$$

This yields the following update equations:

$$S_n = \mathbf{\Sigma}_n + \mathcal{J}(\mathbf{m}_{n-1})\mathbf{P}_{n-1}\mathcal{J}^\top(\mathbf{m}_{n-1}), \tag{73}$$
$$\mathbf{K}_n = \mathbf{P}_{n-1}\mathbf{x}_n^\top \mathcal{J}(\mathbf{m}_{n-1})^\top S_n^{-1}, \tag{74}$$
$$\mathbf{m}_n = \mathbf{m}_{n-1} + \mathbf{K}_n[\mathbf{y}_n - h\left(\mathbf{x}_n\mathbf{m}_{n-1}\right)], \tag{75}$$
$$\mathbf{P}_n = \mathbf{P}_{n-1} - \mathbf{K}_n S_n \mathbf{K}_n^\top. \tag{76}$$

$$S_n = \mathbf{\Sigma}_n + \mathbf{P}_{n-1}\left[\left(1 - \sigma(X_n\mathbf{m}_{n-1})\right)\sigma\left(\mathbf{X}_n\mathbf{m}_{n-1}\right)\right]^2, \tag{77}$$
$$\mathbf{K}_n = \mathbf{P}_{n-1}\mathbf{x}_n^\top \left(1 - \sigma(X_n\mathbf{m}_{n-1})\right)\sigma\left(\mathbf{X}_n\mathbf{m}_{n-1}\right) S_n^{-1}, \tag{78}$$
$$\mathbf{m}_n = \mathbf{m}_{n-1} + \mathbf{K}_n[\mathbf{y}_n - \sigma\left(\mathbf{X}_n\mathbf{m}_{n-1}\right)], \tag{79}$$
$$\mathbf{P}_n = \mathbf{P}_{n-1} - \mathbf{K}_n S_n \mathbf{K}_n^\top. \tag{80}$$

To recover the result in the main paper we may choose to model the process noise $\mathbf{\Sigma}_n \stackrel{\text{def}}{=} \mathbf{x}_n\mathbf{P}_{n-1}\mathbf{x}_n$. This provides a regularization effect by not allowing the update steps to become very large when $\mathbf{x}_n$ has a large magnitude.

Online Multinomial Logistic Regression In the multinomial regression case, the features have the same dimensions, but the observations are a one hot vector $\mathbf{y} \in \mathbb{R}^K$. Here K is the number of classes. Each of the K classes has an associated parameter vector $\theta^{(k)} \in \mathbb{R}^D$, $k \in 1, \ldots, K$. Stacking these vectors yields a parameter matrix $\Theta \in \mathbb{R}^{D\times K}$ with element $\Theta_d^{(k)}$ denoting the weight of input feature at index d for the k-th class.

The function that maps the input vector to the prediction is the softmax function $\hat{p}_l = \frac{\exp({\Theta^{(l)}}^T\mathbf{x})}{1+\sum_{j=1}^K \exp({\Theta^{(j)}}^T\mathbf{x})}$, leading to log likelihood:

$$\mathcal{L}(\Theta) = \left(\sum_{k=1}^K \mathbf{y}_k {\Theta^{(k)}}^T\mathbf{x}\right) - \ln\left(1 + \sum_{j=1}^K \exp({\Theta^{(j)}}^T\mathbf{x})\right). \tag{81}$$

One can show that the gradient of this function evaluates to the following Kronecker product (8):

$$\nabla_\Theta\mathcal{L}(\Theta) = \begin{bmatrix}(\mathbf{y}-\hat{\mathbf{p}})_1\mathbf{x}\\(\mathbf{y}-\hat{\mathbf{p}})_2\mathbf{x}\\\vdots\\(\mathbf{y}-\hat{\mathbf{p}})_K\mathbf{x}\end{bmatrix} = (\mathbf{y}-\hat{\mathbf{p}})\otimes\mathbf{x}, \tag{82}$$

which has the same dimensions as the weight matrix Θ. Thus, in the multinomial case we may still apply the closed form updates derived previously by setting $\mathcal{J}(\Theta) = \nabla_\Theta\mathcal{L}(\Theta)$. This complicates the updates as the parameter is now a matrix but the update complexity remains constant in the size of the parameter and we never have to perform any matrix inversions.

K.2 Relation to other Online Logistic Regression approaches

There are two main other approaches to online logistic regression that are relevant to our method. First, FOLKLORE by Agarwal et al. (2) proposes an iterative optimization scheme where logistic regression parameters can be learned by iteratively solving an optimization problem. While this approach can work well, it lacks the closed form updates that are needed to improve efficiency. An approach with closed form updates is offered in the work of de Vilmarest and Wintenberger (16). These updates are quite similar to ours, and if we remove the regularization from our approach can be made equivalent. However, de Vilmarest and Wintenberger (16) arrive at these updates using Kalman filter theory to derive general results about exponential family models whereas our derivation uses much more basic tools from probability theory. We believe that our approach is intuitive and the derivation makes the linearization assumptions more obvious to the reader rather than automatic application of Kalman filters.

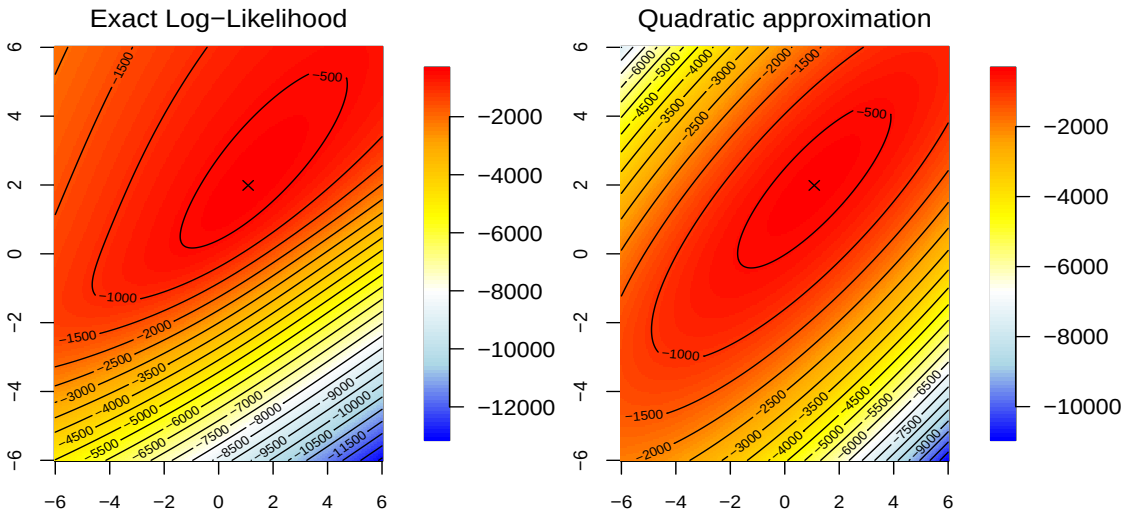

Figure 3: Exact log-likelihood vs. our proposed quadratic approximation. We see that close to the data-generating parameters (marked as x) both our approximation and the exact log-likelihood function agree. See Appendix K.3 for toy dataset experiment details.

K.3 Toy Experiment

In this section we present a toy experiment that analyzes the online logistic regression. We consider the following toy data generating process for a binary logistic regression model with two data generating parameters θ^*:

$$\theta^* = (1, 2) \tag{83}$$

$$\mathbf{X} \sim \mathcal{N}(\mathbf{X} \,|\, -1.5, 1) \tag{84}$$

$$\mathbf{Y} = \begin{cases} 0, & \text{with probability } \frac{1}{1+\exp(\mathbf{x}\cdot\theta)} \\ 1, & \text{with probability } 1 - \frac{1}{1+\exp(\mathbf{x}\cdot\theta)} \end{cases} \tag{85}$$

We generate 100 pairs by drawing independently 100 times from X and generating Y given each sample. Note that $\mathbf{X} \in \mathbb{R}^2$ is augmented with a constant value 1 to handle the intercept parameter. We observe that the log-likelihood shown in Fig. 3 matches well with the quadratic log likelihood approximation around the data generating values.

L Architectural Details

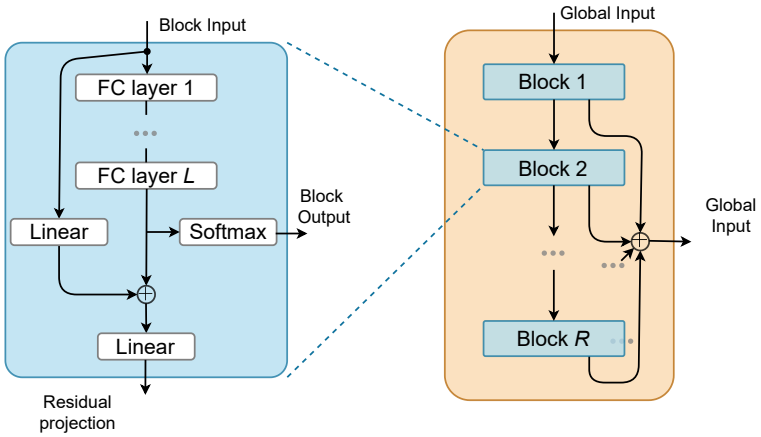

Figure 4: Visualization of our set learning decoder.

Slow learner with set inputs. Rather than masking missing features with a zero or using deterministic dropout, as is done in (3), we treat the input as a set that excludes any missing features. Recall that the data generating process produces a sequence of triplets $\mathcal{T} = \{(\mathbf{z}_1, \mathbf{y}_1, \mathbf{O}_1), \ldots, (\mathbf{z}_T, \mathbf{y}_T, \mathbf{O}_T)\}$. Then, the set of input features $\mathcal{X}_t$ can be expressed as:

$$\mathcal{X}_t = \{\mathbf{z}_{t,j} : \mathbf{O}_{t,j} = 1\}, \quad \mathcal{I}_t = \{j : \mathbf{O}_{t,j} = 1\}, \quad \mathcal{D} = [(\mathcal{X}_1, \mathcal{I}_1, \mathbf{y}_1), \ldots, (\mathcal{X}_T, \mathcal{I}_T, \mathbf{y}_T)]. \tag{86}$$

The size of the input feature set $\mathcal{X}_t$ is time varying. To allow the model to determine which inputs are available, it is necessary to pass a set of feature IDs in an index set $\mathcal{I}_t$. Our proposed set learning module follows closely the ProtoRes architecture (please refer to Oreshkin et al. (38) for details). It takes the index set $\mathcal{I}_t$ and maps each of its active index positions to a continuous representation to create feature ID embeddings. It then concatenates each ID embedding to the corresponding feature value of $\mathcal{X}_t$. These feature values and ID embedding pairs are aggregated and summed to produce fixed dimensional vector representations that we denote as $\mathbf{x}_0$. The main components of our proposed set learning module are blocks, each consisting of L fully connected (FC) layers. Residual skip connections are included in the architecture so that blocks can be bypassed. An input set $\mathcal{X}$ is mapped to $\mathbf{x}_0 = \text{EMB}(\mathcal{X})$. The overall structure of the module at block $r \in \{1, \ldots, R\}$ is:

$$\mathbf{h}_{r,1} = \text{FC}_{r,1}(\mathbf{x}_{r-1}), \quad \ldots, \quad \mathbf{h}_{r,L} = \text{FC}_{r,L}(\mathbf{h}_{r,L-1}), \tag{87}$$

$$\mathbf{x}_r = \text{RELU}(\mathbf{W}_r \mathbf{x}_{r-1} + \mathbf{h}_{r-1,L}), \quad \hat{\mathbf{y}}_r = \hat{\mathbf{y}}_{r-1} + \mathbf{Q}_L \mathbf{h}_{r,L}, \tag{88}$$

where $\mathbf{W}_r, \mathbf{Q}_L$ are learnable matrices. We connect R blocks sequentially to obtain global output $\hat{\mathbf{y}}_R$.

M Fast Online Deep Learning - Additional Details

M.1 Proof of Proposition 2

The proof is an inductive argument. First, we analyze a base case for a $L = 3$ layer network for a proof sketch.

BASE CASE: Prove the proposition holds for $L = 3$.

Recall the variable definitions:

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

And the update rule:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \sum_{j=l}^{L} \alpha^{(j)} \nabla_{W^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t) \tag{89}$$

Then for each layer we get the following updates (Fig. 5 shows the ODL computational graph):

$$\mathbf{W}_{t+1}^{(3)} = \mathbf{W}_t^{(3)} - \eta \nabla_{\mathbf{W}_t^{(3)}} \alpha^{(3)} \mathcal{L}_3 = \mathbf{W}_t^{(3)} - \eta \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{W}_t^{(3)}} \tag{90}$$

$$\begin{aligned} \mathbf{W}_{t+1}^{(2)} &= \mathbf{W}_t^{(2)} - \eta \left(\nabla_{\mathbf{W}_t^{(2)}} \alpha^{(2)} \mathcal{L}_2 + \nabla_{\mathbf{W}_t^{(2)}} \alpha^{(3)} \mathcal{L}_3 \right) \\ &= \mathbf{W}_t^{(2)} - \eta \left(\alpha_2 \frac{\partial \mathcal{L}_2}{\partial \mathbf{f}^{(2)}} \frac{\partial \mathbf{f}^{(2)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}_t^{(2)}} + \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}_t^{(2)}} \right) \end{aligned} \tag{91}$$

$$\begin{aligned} \mathbf{W}_{t+1}^{(1)} &= \mathbf{W}_t^{(1)} - \eta \left(\nabla_{\mathbf{W}_t^{(1)}} \alpha^{(1)} \mathcal{L}_1 + \nabla_{\mathbf{W}_t^{(1)}} \alpha^{(2)} \mathcal{L}_2 + \nabla_{\mathbf{W}_t^{(1)}} \alpha^{(3)} \mathcal{L}_3 \right) \\ &= \mathbf{W}_t^{(1)} - \eta \left(\alpha_1 \frac{\partial \mathcal{L}_1}{\partial \mathbf{f}^{(1)}} \frac{\partial \mathbf{f}^{(1)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} + \alpha_2 \frac{\partial \mathcal{L}_2}{\partial \mathbf{f}^{(2)}} \frac{\partial \mathbf{f}^{(2)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} + \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} \right) \end{aligned} \tag{92}$$

Now, we consider our proposed architecture, shown in Fig. 6. In our case the weights can be updated using standard backpropagation, i.e., $\mathbf{W}_{t+1}^{(l)} \leftarrow \mathbf{W}_t^{(l)} - \eta \nabla_{\mathbf{W}_t^{(l)}} \mathcal{L}$, where $\mathcal{L} = \sum_{i=1}^{L} \alpha^{(i)} \mathcal{L}_i$.

Lemma 1: The gradient $\frac{\partial \mathcal{L}_l}{\partial \mathbf{W}^{(k)}} = 0$ for all $(k, l) \in \{1, 2, \dots, L\}^2$ with $k > l$.

Proof: Suppose $k > l$ and recall from the definition, $\mathbf{f}^{(l)} = \text{softmax}(\mathbf{h}^{(l)}\Theta^{(l)})$, with $\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)})$. Now following the recursive definition of $\mathbf{h}^{(l)}$ we have:

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)}\right) = \sigma\left(\mathbf{W}^{(l)}\sigma\left(\mathbf{W}^{(l-1)}\sigma\left(\cdots\left(\sigma\left(\mathbf{W}^{(1)}\mathbf{h}^{(0)}\right)\right)\cdots\right)\right)\right) \tag{93}$$

$$\mathbf{f}^{(l)} = \text{softmax}(\mathbf{h}^{(l)}\Theta^{(l)}) = \text{softmax}\left[\sigma\left(\mathbf{W}^{(l)}\sigma\left(\mathbf{W}^{(l-1)}\sigma\left(\cdots\left(\sigma\left(\mathbf{W}^{(1)}\mathbf{h}^{(0)}\right)\right)\cdots\right)\right)\right)\mathbf{\Theta}^{(l)}\right] \tag{94}$$

Observe that for $k > l$, $\mathbf{W}^{(k)}$ does not appear in eq. (94) thus we immediately conclude $\frac{\partial \mathbf{f}^{(l)}}{\partial \mathbf{W}^{(k)}} = 0$. Recall that the operator $\mathcal{L}_l$ is a function of $\mathbf{f}^{(l)}$, *i.e.*, $\mathcal{L}_l(\mathbf{f}^{(l)}, y^{(t)})$. Given that $y^{(t)}$ is a fixed constant, by the chain rule, $\frac{\partial \mathcal{L}_l}{\partial \mathbf{W}^{(k)}} = \frac{\partial \mathcal{L}_l}{\partial \mathbf{f}^{(l)}} \frac{\partial \mathbf{f}^{(l)}}{\partial \mathbf{W}^{(k)}} = 0$.

Corollary: 1 $\nabla_{\mathbf{W}_t^{(k)}} \mathcal{L} = \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=1}^{L} \alpha^{(i)} \mathcal{L}_i = \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=k}^{L} \alpha^{(i)} \mathcal{L}_i$.

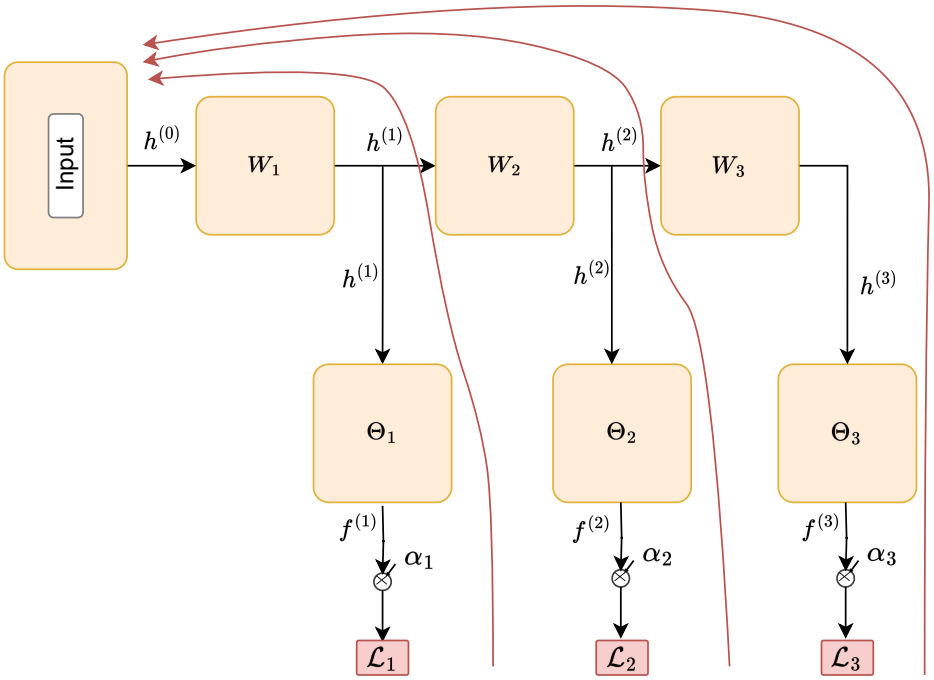

Figure 5: ODL hedge backpropagation. Red lines indicate individual backpropagation calculations.

Proof: All $i < k$ the terms in the sum are equal to zero by direct application of the previous lemma:

$$\begin{aligned}\nabla_{\mathbf{W}_t^{(k)}} \mathcal{L} &= \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=1}^{L} \alpha^{(i)} \mathcal{L}_i = \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=1}^{k-1} \alpha^{(i)} \mathcal{L}_i + \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=k}^{L} \alpha^{(i)} \mathcal{L}_i \\ &= \underbrace{\nabla_{\mathbf{W}_t^{(k)}} \alpha^{(1)} \mathcal{L}_1}_{=0 \text{ by lemma 1}} + \cdots + \underbrace{\nabla_{\mathbf{W}_t^{(k)}} \alpha^{(k-1)} \mathcal{L}_{k-1}}_{=0 \text{ by lemma 1}} + \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=k}^{L} \alpha^{(i)} \mathcal{L}_i \\ &= \nabla_{\mathbf{W}_t^{(k)}} \sum_{i=k}^{L} \alpha^{(i)} \mathcal{L}_i \,. \end{aligned} \tag{95}$$

Now, using Lemma 1 and by linearity of differentiation we have:

$$\begin{aligned}\mathbf{W}_{t+1}^{(3)} &= \mathbf{W}_t^{(3)} - \eta \nabla_{\mathbf{W}_t^{(3)}} \mathcal{L} = \mathbf{W}_t^{(3)} - \eta \nabla_{\mathbf{W}_t^{(3)}} \sum_{i=1}^{3} \alpha^{(i)} \mathcal{L}_i = \mathbf{W}_t^{(3)} - \eta \nabla_{\mathbf{W}_t^{(3)}} \alpha^{(3)} \mathcal{L}_3 \\ &= \mathbf{W}_t^{(3)} - \eta \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{W}_t^{(3)}} \end{aligned} \tag{96}$$

$$\begin{aligned}\mathbf{W}_{t+1}^{(2)} &= \mathbf{W}_t^{(2)} - \eta \nabla_{\mathbf{W}_t^{(2)}} \mathcal{L} = \mathbf{W}_t^{(2)} - \eta \nabla_{\mathbf{W}_t^{(2)}} \sum_{i=1}^{3} \alpha^{(i)} \mathcal{L}_i = \mathbf{W}_t^{(2)} - \eta \nabla_{\mathbf{W}_t^{(2)}} \left(\alpha^{(2)} \mathcal{L}_2 + \alpha^{(3)} \mathcal{L}_3 \right) \\ &= \mathbf{W}_t^{(2)} - \eta \left(\alpha_2 \frac{\partial \mathcal{L}_2}{\partial \mathbf{f}^{(2)}} \frac{\partial \mathbf{f}^{(2)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}_t^{(2)}} + \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}_t^{(2)}} \right) \end{aligned} \tag{97}$$

$$\begin{aligned}\mathbf{W}_{t+1}^{(1)} &= \mathbf{W}_t^{(1)} - \eta \nabla_{\mathbf{W}_t^{(1)}} L = \mathbf{W}_t^{(1)} - \eta \nabla_{\mathbf{W}_t^{(1)}} \sum_{i=1}^{3} \alpha^{(i)} \mathcal{L}_i = \mathbf{W}_t^{(1)} - \eta \nabla_{\mathbf{W}_t^{(1)}} \left(\alpha^{(1)} \mathcal{L}_1 + \alpha^{(2)} \mathcal{L}_2 + \alpha^{(3)} \mathcal{L}_3 \right) \\ &= \mathbf{W}_t^{(1)} - \eta \left(\alpha_1 \frac{\partial \mathcal{L}_1}{\partial \mathbf{f}^{(1)}} \frac{\partial \mathbf{f}^{(1)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} + \alpha_2 \frac{\partial \mathcal{L}_2}{\partial \mathbf{f}^{(2)}} \frac{\partial \mathbf{f}^{(2)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} + \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} \right) \end{aligned} \tag{98}$$

The above 3 equations show that for $L = 3$ and $l \in \{1, 2, 3\}$:

$$\mathbf{W}_t^{(l)} - \eta \sum_{j=l}^{L} \alpha^{(j)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t) = \mathbf{W}_t^{(l)} - \eta \nabla_{\mathbf{W}^{(l)}} \sum_{j=1}^{L} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t)\,. \tag{99}$$

INDUCTIVE HYPOTHESIS: Assume that the proposition holds for $L = K$, $l \in \{1, \ldots, L\}$.

$$\mathbf{W}_t^{(l)} - \eta \sum_{j=l}^{K} \alpha^{(j)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t) = \mathbf{W}_t^{(l)} - \eta \nabla_{\mathbf{W}^{(l)}} \sum_{j=1}^{K} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t)\,. \tag{100}$$

INDUCTIVE STEP: Prove that the proposition holds for $L = K + 1$, $l \in \{1, \ldots, L\}$.

We want to show

$$\mathbf{W}_t^{(l)} - \eta \sum_{j=l}^{K+1} \alpha^{(j)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t) = \mathbf{W}_t^{(l)} - \eta \nabla_{\mathbf{W}^{(l)}} \sum_{j=1}^{K+1} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t)\,. \tag{101}$$

Consider the left hand side of the above equation. And apply inductive hypothesis on first K terms of sum.

$$\mathbf{W}_t^{(l)} - \eta \sum_{j=l}^{K+1} \alpha^{(j)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t) = \mathbf{W}_t^{(l)} - \eta \left[\sum_{j=l}^{K} \alpha^{(j)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(j)}, y_t) \right] - \eta \alpha^{(l)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(K+1)}, y_t) \tag{102}$$

$$= \mathbf{W}_t^{(l)} - \eta \nabla_{\mathbf{W}^{(l)}} \sum_{j=1}^{K} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t) - \eta \alpha^{(K+1)} \nabla_{\mathbf{W}^{(l)}} \mathcal{L}(\mathbf{f}^{(K+1)}, y_t) \tag{103}$$

$$= \mathbf{W}_t^{(l)} - \eta \nabla_{\mathbf{W}^{(l)}} \sum_{j=1}^{K+1} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t)\,. \tag{104}$$

This closes the induction step and completes the proof.

M.2 Complexity analysis

In this section we compare ODL and our proposed modified optimization setup, which we call Fast-ODL (FODL), in terms of training time complexity.

ODL training complexity: Consider the computations required to perform an update on hidden layer parameters $\mathbf{W}^{(l)}$. Based on the update rule eq. (89), to update weight matrix $\mathbf{W}^{(l)}$ we need to calculate $L - l + 1$ terms in the sum. Assuming that calculating each partial derivative in the previous expressions takes constant time, and noting that the weight update needs to be done for each layer $l \in \{0, 1, \ldots, L\}$ we obtain a complexity of:

$$\sum_{j=1}^{L} \sum_{i=j}^{L} O(1) = \frac{L(L+1)}{2} = O(L^2) \tag{105}$$

per training step. Given that we are in an online learning setting with batch size of 1, in a dataset of size n we are thus going to take n training steps (one step per training datum) for a total training complexity of:

$$\sum_{1}^{n} \sum_{j=1}^{L} \sum_{i=j}^{L} O(1) = O(nL^2) \tag{106}$$

The quadratic complexity in the number of layers occurs because of redundant calculations. For example, to update both $\mathbf{W}^{(l)}$ and $\mathbf{W}^{(l-1)}$ we need to compute $\frac{\partial \mathcal{L}_l}{\partial \mathbf{f}^{(l)}}$. However, in the vanilla ODL setup this result is not cached and needs to be recomputed two separate times.

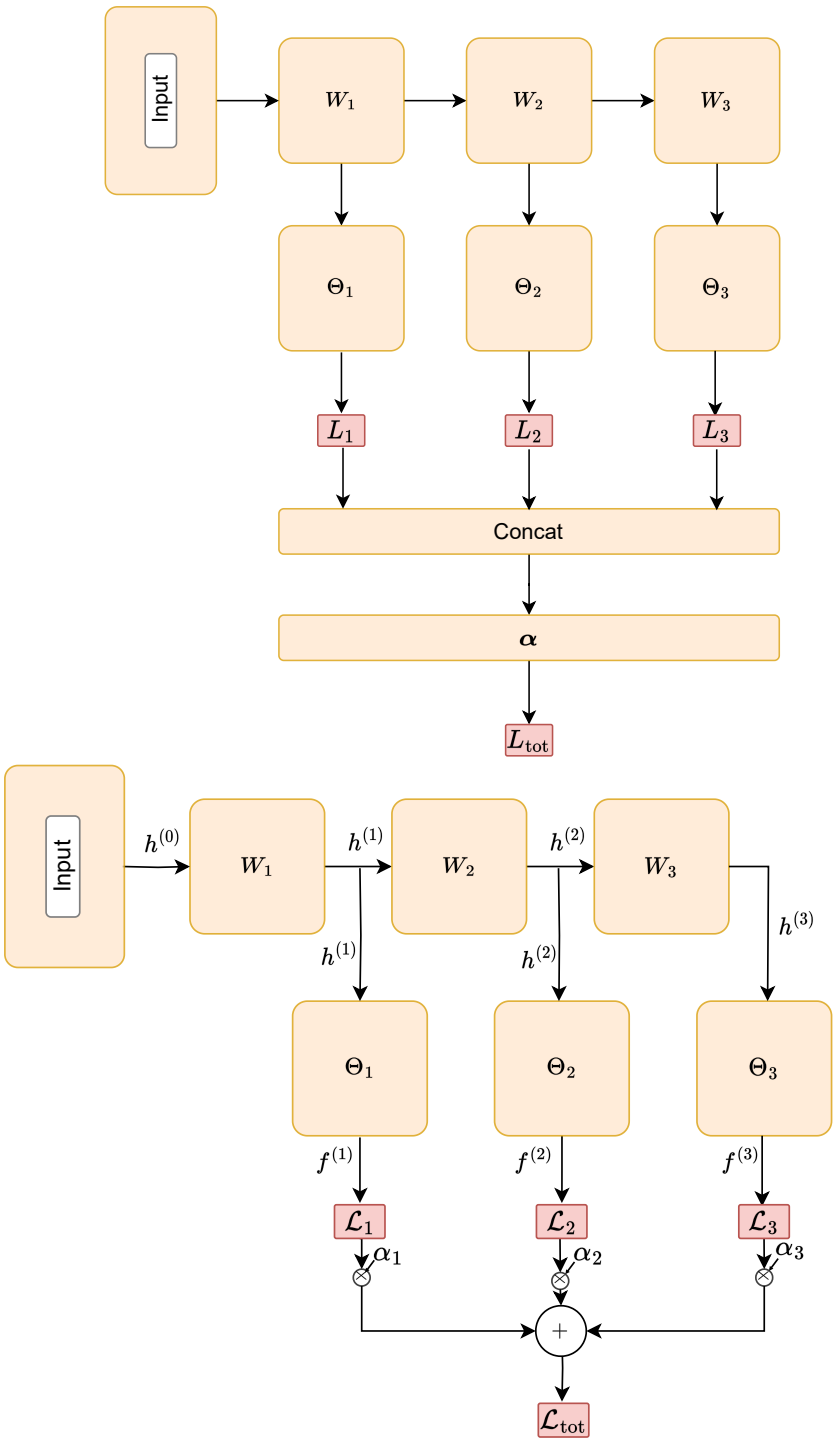

Figure 6: Our proposed architectural modification that provides equivalent total gradients but a more tractable computational graph. There is only one gradient calculation starting from $\mathcal{L}_{\text{tot}}$. Top and bottom figure show the same architecture, bottom emphasizes that α_i is a constant that is not optimized during backprop.

FODL training complexity: Looking at eq. (89), we note that the number of gradient calculations would decrease significantly if the the gradient could be taken outside the sum:

$$W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta \nabla_{W^{(l)}} \sum_{j=l}^{L} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t). \tag{107}$$

However, this expression cannot be directly computed using automatic differentiation as there is no node equal to $\sum_{j=l}^{L} \alpha^{(j)} \mathcal{L}(\mathbf{f}^{(j)}, y_t)$ is the computational graph in Fig 5. By introducing a concatenation and weighted summation of the intermediate losses, as shown in Fig. 6, we can generate this node and backpropagate from it. Then using PyTorch's automatic differentiation framework, computed gradients are cached such that there is no redundant calculation, *i.e.*, when updating $\mathbf{W}^{(1)}$ we compute $\frac{\partial \mathcal{L}_l}{\partial \mathbf{f}^{(l)}}$ and store it. Then for all subsequent calculations for updating $\mathbf{W}^{(l)}$ we access the cache at negligible computational cost. This yields a total training cost of:

$$\sum_{1}^{n} \sum_{j=1}^{L} nO(1) = O(nL). \tag{108}$$

For example, consider computing $\mathbf{W}_{t+1}^{(3)}$ after having computed $\mathbf{W}_{t+1}^{(1)}$. Recalling the equations for their updates shown earlier in this section, we know that:

$$\begin{aligned}
\mathbf{W}_{t+1}^{(3)} &= \mathbf{W}_t^{(3)} - \eta\alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{W}_t^{(3)}} \\
\mathbf{W}_{t+1}^{(2)} &= \mathbf{W}_t^{(2)} - \eta \left(\alpha_2 \frac{\partial \mathcal{L}_2}{\partial \mathbf{f}^{(2)}} \frac{\partial \mathbf{f}^{(2)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}_t^{(2)}} + \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}_t^{(2)}} \right) \\
\mathbf{W}_{t+1}^{(1)} &= \mathbf{W}_t^{(1)} - \eta \Bigg(\alpha_1 \frac{\partial \mathcal{L}_1}{\partial \mathbf{f}^{(1)}} \frac{\partial \mathbf{f}^{(1)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} + \alpha_2 \frac{\partial \mathcal{L}_2}{\partial \mathbf{f}^{(2)}} \frac{\partial \mathbf{f}^{(2)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} \\
&\qquad + \alpha_3 \frac{\partial \mathcal{L}_3}{\partial \mathbf{f}^{(3)}} \frac{\partial \mathbf{f}^{(3)}}{\partial \mathbf{h}^{(3)}} \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}_t^{(1)}} \Bigg)
\end{aligned}$$

Clearly, caching the term in teal would reduce computation. Thus, in FODL computing the update for the first layer $\mathbf{W}_{t+1}^{(1)}$ and storing all the partial derivatives along the computational graph provides all the necessary information to calculate $\mathbf{W}_{t+1}^{(2)}, \ldots, \mathbf{W}_{t+1}^{(L)}$ with computing only one new derivative $\frac{\partial \mathbf{h}^{(l)}}{\partial \mathbf{W}_t^{(l)}}$. Similarly this holds for computing $\mathbf{W}_{t+1}^{(2)}$ after $\mathbf{W}_{t+1}^{(1)}$, see orange terms.

More precisely, in the standard ODL formulation the redundancy of the teal term is very high as it is computed L times (the terms in orange $L-1$ times) but in our formulation every gradient is only computed once. This yields a total computational saving of:

$$\sum_{1}^{n} \sum_{j=1}^{L} \sum_{i=j+1}^{L} = n \frac{L(L-1)}{2} O(1). \tag{109}$$

By subtracting the amount of computation saved by FODL from the total complexity of the base ODL method we obtain the complexity of our proposed approach and it is linear:

$$\sum_{1}^{n} \sum_{j=1}^{L} \sum_{i=j}^{L} O(1) - \sum_{1}^{n} \sum_{j=1}^{L} \sum_{i=j+1}^{L} O(1) = n \left(\frac{L(L+1)}{2} - \frac{L(L-1)}{2} \right) O(1) = nLO(1) = O(nL). \tag{110}$$