Knowledge-Adaptation Priors

Anonymous Author(s)
Affiliation
Address
email

Abstract

Humans and animals have a natural ability to quickly adapt to their surroundings, but machine-learning models, when subjected to changes in their training setup, often require a complete retraining from scratch. We present Knowledge-adaptation priors (K-priors) to reduce the cost of retraining by enabling *quick and accurate adaptation for a wide-variety of tasks and models*. This is made possible by combining weight and function-space priors to reconstruct the gradients of the past, which surprisingly recovers and generalizes many existing, but seemingly-unrelated, adaptation strategies. Training with simple first-order gradient methods can often recover the exact retrained model to an arbitrary accuracy by choosing a sufficiently large memory of the past data. Empirical results confirm that the adaptation can be cheap and accurate, and a promising alternative to retraining.

1 Introduction

Machine-Learning (ML) at production often requires constant model updating which can have huge financial and environmental costs [16, 36]. The production pipeline is continuously evolving, where new data are regularly pooled and labeled and old data become irrelevant. Regular tuning of hyperparameters is required to handle drifts [16], and sometimes even the model class/architecture may need to change. Due to this, the model is frequently retrained, retested, and redeployed, which can be extremely costly, especially when the data and model sizes are large. The cost can be reduced if, instead of repeatedly retraining from scratch, the system can quickly adapt to small changes. Humans and animals can naturally use their prior knowledge to handle a wide-variety of changes in their surroundings, but such *quick, wide, and accurate adaptation* has been difficult to achieve in ML.

In theory, this should be possible within a Bayesian framework where the posterior is used as the prior for the future, but exact Bayes is computationally challenging and designing generic Bayesian priors has its own challenges [46]. In ML, simpler mechanisms are more popular, e.g., in Support Vector Machines (SVMs) for adding/removing data [9, 49, 51], and in deep learning for model compression [21]. Weight-priors are used in online learning [10], and more recently for continual learning [24, 33, 39, 27, 53], but they are not suited for many other adaptation tasks (e.g., model compression). In some settings, they also perform worse, e.g., in continual learning when compared to memory-based strategies [28, 38, 37, 6]. All these previous works in ML apply to narrow, specific settings, and designing generic adaptation-mechanisms remains an open challenge.

We present Knowledge-adaptation priors (K-priors) for designing generic adaptation-mechanisms for ML applications. The general principle of adaptation, we propose through K-priors, is to combine the weight and function-space divergences to faithfully reconstruct the gradient of the past. This gives rise to adaptation mechanisms that apply to a wide-variety of tasks (Fig. 1, left) and models (e.g., generalized linear models, deep networks, and their Bayesian extensions), and also connect to many seemingly-unrelated existing works in specific settings (e.g., weight-priors [10], knowledge distillation [21], SVMs [9, 29, 49], online Gaussian Processes [14], and continual learning [30, 37, 6]). The principle leads to natural adaptation-mechanisms where models' predictions need to be readjusted

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

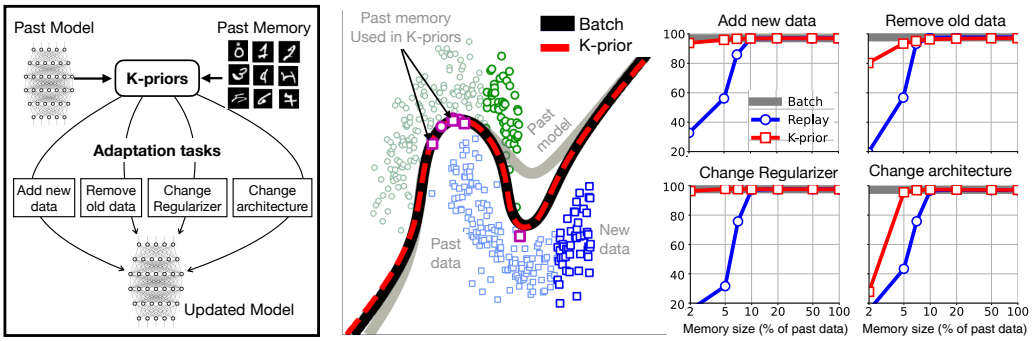

Figure 1: **Left:** K-priors can handle a wide-variety of adaptation tasks by using the past model and a small memory of the past data. **Middle:** For adaptation, the model needs tweaking at only a handful of past examples (shown with dark purple markers). The adapted model (dashed red line) is very close to the retraining on the full batch (solid black line). **Right:** Results on binary classification on 'USPS' digits with neural networks show that K-priors (red square) obtain solutions close to the batch training (gray line) but by only using a small fraction of the past data. It also performs better than 'Replay' (blue circles) where the same memory is used for replay. See Section 5 for more details.

only at a handful of past experiences (see Fig. 1, middle). This is quick and easy to train with first-order optimization methods, and by choosing a sufficiently large memory, we get results similar to those obtained by retraining from scratch (see Fig. 1, right). Even though derived differently, our proposal aligns well the general principles of constructing priors [46], giving us a class of informative K-priors which are computationally feasible, and aware of the problem structure and changes to it.

2 Adaptation Tasks in Machine Learning

Our goal is to quickly and accurately adapt to a wide range of changes in the training setup. We denote the *base model* by $f_{\mathbf{w}_*}(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^D$, weights $\mathbf{w}_* \in \mathbb{R}^P$, and data $\mathcal{D}$, by solving

$$\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathcal{W}} \; \bar{\ell}(\mathbf{w}), \qquad \text{where } \bar{\ell}(\mathbf{w}) = \sum_{i\in\mathcal{D}} \ell_i(\mathbf{w}) + \mathcal{R}(\mathbf{w}). \tag{1}$$

Here, $\ell_i(\mathbf{w})$ denotes the loss function on the i'th data example, and $\mathcal{R}(\mathbf{w})$ is a regularizer. Our goal is to adapt $\mathbf{w}_*$ to small changes in this training setup to reduce the cost of retraining from scratch. We are interested in methods that can handle a wide-variety of changes. The simplest and most common change is to add/remove data examples, as shown below at the left where the data example $j \notin \mathcal{D}$ is added to get $\mathbf{w}_+$ and at the right where an example $k \in \mathcal{D}$ is removed to get $\mathbf{w}_-$,

$$\mathbf{w}_+ = \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{D}\cup j} \ell_i(\mathbf{w}) + \mathcal{R}(\mathbf{w}), \qquad \mathbf{w}_- = \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{D}\setminus k} \ell_i(\mathbf{w}) + \mathcal{R}(\mathbf{w}). \tag{2}$$

We refer to these as 'Add/Remove Data' tasks. Multiple examples can be added/removed.

Another task is to adapt to the change in the regularizer or its hyperparameter (see left, where the new regularizer is denoted by $\mathcal{G}(\mathbf{w})$, e.g., changing the hyperparameter of the L_2 prior) or even change the model class/architecture (see right, where the optimization is in the new parameter space Θ),

$$\mathbf{w}_{\mathcal{G}} = \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i\in\mathcal{D}} \ell_i(\mathbf{w}) + \mathcal{G}(\mathbf{w}), \qquad \boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i\in\mathcal{D}} \tilde{\ell}_i(\boldsymbol{\theta}) + \tilde{\mathcal{R}}(\boldsymbol{\theta}). \tag{3}$$

Here, we assume $\tilde{\ell}_i(\boldsymbol{\theta})$ to have the same form as $\ell_i(\mathbf{w})$ but use the new model $\tilde{f}_{\boldsymbol{\theta}}(\mathbf{x})$, where $\boldsymbol{\theta} \in \Theta$ is the new parameter, possibly of a different dimension. The new regularizer could be chosen accordingly. Changing the model class could be a simple change in features, e.g., in linear models where features are added/removed, or model compression as in knowledge distillation [21] in which case the regularizer may not be required. We will also mention the addition of privileged information, originally proposed in [51], where the goal is to include additional inputs/output pairs to improve the performance of a model. This is unified with knowledge distillation by [31] and recently been applied for domain adaptation in deep learning [2, 41, 42].

Other tasks could include a change in the loss function, or even a combination of a few of these. k-fold Cross-validation is perhaps one of the simplest applications, where in each run we need to remove an old subset of data, add a new one, and then change the hyperparameters over a pre-specified grid. We can have model selection or architecture search on top of such procedures. For real-world ML tasks, a quick and accurate adaptation could lead to a huge reduction in the cost of retraining.

Most of the previous work apply to one or two of these specific adaptation tasks, and there is almost no work on the adaptation-mechanisms that apply more widely. For SVMs, a lot of work has focused on adding/removing data [9, 50, 22, 17, 40, 29, 25, 49]. Many recent proposals for deleting data by using machine-*unlearning* [8, 19, 34, 20, 44] have been considered due to privacy concerns [18]. Weight-priors, used in online/continual learning, too have focused mostly on adding data and has not been applied for other tasks, possibly because they are not easily amenable to such changes. The same is true for functional regularizers for continual learning [28, 38, 5, 48, 37, 6], although some of these works are indeed inspired by knowledge distillation [28, 6]. It is not clear how they can handle changes in the weight regularizers, for example, if we want to use a previously trained model with L_2 prior to train a sparse network that uses an L_1 prior. Online learning in Gaussian-Process (GP) models [14, 43, 47] has also largely focused on the 'Add/Remove Data' tasks.

To the best of our knowledge, there is no prior work on the 'Hyperparameter/Regularizer Change' task. In SVMs, adaptation has been used to boost hyperparameter tuning [52] but only 'Add/Remove Data' is employed, along with some warm-start techniques [15]. Such *warm-starting* is often not sufficient when the new solution is quite far from the old one, and can even hurt performance [3].

Finally, even though all these works focus on the same task of adding/removing data, their working methods are entirely different from each other, and they do not appear to have any common principles to describe their adaptation mechanisms. Our goal is to fill this gap and propose a single method that can handle a wide-variety of tasks for a range of models, through one general principle of adaptation.

Throughout, we will use a supervised problem. Extension to general learning frameworks is in the last paragraph of Section 3. We will assume that the loss is specified by an exponential-family,

$$\ell(y, h(f)) = -\log p(y|f) = -\langle y, f\rangle + A(f), \tag{4}$$

with $y \in \mathcal{Y}$ as the output, $f \in \mathcal{F}$ as the canonical natural parameter, $A(f)$ as the log-partition function, and $h(f) = \mathbb{E}(y) = \nabla A(f)$ as the expectation parameter (predictions). A typical example is the cross-entropy loss where $A(f) = \log(1 + e^f)$ and $h(f) = \sigma(f)$ is the Sigmoid function. The definition applies to multi-output functions too. We will use a shorthand for the model outputs denoting $f^i_{\mathbf{w}} = f_{\mathbf{w}}(\mathbf{x}_i)$ and repeatedly make use of the following derivative of the loss,

$$\nabla \ell(y_i, h(f^i_{\mathbf{w}})) = \nabla f^i_{\mathbf{w}} \; [h(f^i_{\mathbf{w}}) - y_i]. \tag{5}$$

3 Knowledge-Adaptation Priors (K-priors)

We present Knowledge-adaptation priors for designing generic adaptation-mechanisms to handle wide-variety of tasks for a range of models. By K-priors, denoted by $\mathcal{K}(\mathbf{w})$, we refer to a class of regularizers that penalize a candidate weight-vector $\mathbf{w}$ by measuring the discrepancies in both the weight and function-spaces by using divergence functions $\mathbb{D}_w$ and $\mathbb{D}_f$ respectively,

$$\mathcal{K}(\mathbf{w}; \mathbf{w}_*, \mathcal{M}) = \mathbb{D}_f\left(\mathbf{f}(\mathbf{w}) \| \mathbf{f}(\mathbf{w}_*)\right) + \tau \mathbb{D}_w\left(\mathbf{w} \| \mathbf{w}_*\right). \tag{6}$$

Here, $\mathbf{f}(\mathbf{w})$ refers to a vector of $f_{\mathbf{w}}(\mathbf{u}_i)$ at M fixed inputs $\mathbf{u}_i \in \mathcal{M}$ (we also allow $\mathbf{u}_i \notin \mathcal{D}$). The divergence $\mathbb{D}_f(\cdot\|\cdot)$ measures the distance between $f_{\mathbf{w}}(\mathbf{u}_i)$ and $f_{\mathbf{w}_*}(\mathbf{u}_i)$ in the function space $\mathcal{F}$, while $\mathbb{D}_w(\cdot\|\cdot)$ measures the same but in the weight space $\mathcal{W}$. Throughout, we will use Bregman divergences $\mathcal{B}_\psi(p_1, p_2) = \psi(p_1) - \psi(p_2) - \nabla\psi(p_2)(p_1 - p_2)$ given a convex function $\psi(\cdot)$.

The set of parameters of a K-prior consists of the *base model* $\mathbf{w}_*$, the *memory set* $\mathcal{M}$, and $\tau > 0$, but could also contain other parameters required to define the divergence functions. The dependency on the parameters will generally be omitted for notational convenience, and we may use $\mathcal{K}(\mathbf{w})$ instead.

Our general principle of adaptation is to use $\mathcal{K}(\mathbf{w})$ to faithfully reconstruct the gradients of the past training objective (1). We show that this is generally possible by combining the two divergences. We now illustrate this for Generalized Linear Models (GLMs) then discuss extensions in Section 4.

K-priors for GLMs. For GLMs, such as logistic and Poisson regression, we have $f^i_{\mathbf{w}} = \boldsymbol{\phi}_i^\top \mathbf{w}$ where $\boldsymbol{\phi}_i = \boldsymbol{\phi}(\mathbf{x}_i)$ is a P-dimensional basis-function vector. For illustration purposes, we assume an L_2

regularizer $\mathcal{R}(\mathbf{w}) = \frac{1}{2}\delta\|\mathbf{w}\|^2$ with $\delta > 0$, but K-priors can be defined for any regularizers. We define K-priors with weight-space divergence as the Bregman divergence generated with $\mathcal{R}(\mathbf{w})$,

$$\mathbb{D}_w\left(\mathbf{w}\|\mathbf{w}_*\right) = \mathcal{B}_{\mathcal{R}}(\mathbf{w}\|\mathbf{w}_*) = \tfrac{1}{2}\delta\|\mathbf{w}-\mathbf{w}_*\|^2,$$

and function-space divergence is chosen as the Bregman divergence with $A(f)$ of the loss (4), with $\mathcal{M} = \mathcal{D}$ and separated across the data examples (smaller memories are discussed later in this section),

$$\mathbb{D}_f\left(\mathbf{f}(\mathbf{w})\|\mathbf{f}(\mathbf{w}_*)\right) = \sum_{i\in\mathcal{D}} \mathcal{B}_A(f^i_{\mathbf{w}}\|f^i_{\mathbf{w}_*}) = \sum_{i\in\mathcal{D}} \ell\left(h(f^i_{\mathbf{w}_*}), h(f^i_{\mathbf{w}})\right) + \text{constant}.$$

By choosing $\tau = 1$, we get the following K-prior,

$$\mathcal{K}(\mathbf{w}) = \sum_{i\in\mathcal{D}} \ell\left(h(f^i_{\mathbf{w}_*}), h(f^i_{\mathbf{w}})\right) + \tfrac{1}{2}\delta\|\mathbf{w}-\mathbf{w}_*\|^2, \tag{7}$$

which has a similar form to (1), but the outputs y_i are now replaced by the predictions $h(f^i_{\mathbf{w}_*})$, and the base model $\mathbf{w}_*$ serves as the mean of a Gaussian weight prior. K-priors use both the Euclidean distance in the weight-space and the prediction discrepancies in the function-space.

Remarkably, the gradient of the above K-prior is exactly equal to that of $\bar{\ell}(\mathbf{w})$ used to obtain $\mathbf{w}_*$. This is shown below, where the first line is obtained by using (5) and noting that $\nabla f^i_{\mathbf{w}} = \boldsymbol{\phi}_i$, and the second line is obtained by adding and subtracting outputs y_i in the first term,

$$\nabla\mathcal{K}(\mathbf{w}) = \sum_{i\in\mathcal{D}} \boldsymbol{\phi}_i\left[h(f^i_{\mathbf{w}}) - h(f^i_{\mathbf{w}_*})\right] + \delta(\mathbf{w}-\mathbf{w}_*), \tag{8}$$

$$= \underbrace{\sum_{i\in\mathcal{D}} \boldsymbol{\phi}_i\left[h(f^i_{\mathbf{w}}) - y_i\right] + \delta\mathbf{w}}_{=\nabla\bar{\ell}(\mathbf{w}).} \;-\; \underbrace{\cancel{\sum_{i\in\mathcal{D}} \boldsymbol{\phi}_i\left[h(f^i_{\mathbf{w}_*}) - y_i\right] - \delta\mathbf{w}_*}}_{=0.}\,. \tag{9}$$

The second term is equal to 0 because $\mathbf{w}_*$ is a minimizer and therefore $\nabla\bar{\ell}(\mathbf{w}_*) = 0$. In this case, the K-prior with $\mathcal{M} = \mathcal{D}$ exactly recovers the gradient (and also Hessian) of the past training objective.

Adaptation for GLMs with K-priors. Since the K-prior above exactly recovers the gradient of $\bar{\ell}(\mathbf{w})$, we can use it to train the next task instead of retraining from scratch. For example, for the 'Add/Remove Data' tasks, we can use the following K-prior regularized objectives,

$$\hat{\mathbf{w}}_+ = \arg\min_{\mathbf{w}\in\mathcal{W}}\; \ell_j(\mathbf{w}) + \mathcal{K}(\mathbf{w}), \qquad \hat{\mathbf{w}}_- = \arg\min_{\mathbf{w}\in\mathcal{W}}\; -\ell_k(\mathbf{w}) + \mathcal{K}(\mathbf{w}). \tag{10}$$

Using (9), it is easy to show that this recovers the retraining-from-scratch solution.

Theorem 1. *For GLMs with a strictly convex $\mathcal{R}(\mathbf{w})$, we have $\mathbf{w}_+ = \hat{\mathbf{w}}_+$ and $\mathbf{w}_- = \hat{\mathbf{w}}_-$.*

We stress that (10) is fundamentally different from full retraining (2) because *here we do not require the true outputs y_i*, but only the predictions of $\mathbf{w}_*$. It is surprising then that we can still recover the exact solution. Since we do not need labels, *any* input locations, even those outside the training data, can be used as long as they closely approximate the gradient. The memory set $\mathcal{M}$ therefore can be a few representative inputs which approximate the gradients well enough.

To change the regularizer or its hyperparameter, we need to replace the weight-space divergence in (7) with a Bregman divergence defined using two different regularizers ([35]; see Proposition 5),

$$\mathcal{B}_{\mathcal{GR}}(\mathbf{w}\|\mathbf{w}_*) = \mathcal{G}(\mathbf{w}) + \mathcal{R}^*(\boldsymbol{\eta}_*) - \mathbf{w}^\top\boldsymbol{\eta}_*, \tag{11}$$

where $\boldsymbol{\eta}_* = \nabla\mathcal{R}(\mathbf{w}_*)$ is the dual parameter and $\mathcal{R}^*$ is the convex-conjugate of $\mathcal{R}$. This is very similar to the standard Bregman divergence but uses two different (convex) generating functions. To get an intuition, consider hyperparameter-tuning for the L_2 regularizer $\mathcal{R}(\mathbf{w}) = \frac{1}{2}\delta\|\mathbf{w}\|^2$, where our new regularizer $\mathcal{G}(\mathbf{w}) = \frac{1}{2}\gamma\|\mathbf{w}\|^2$ uses a hyperparameter $\gamma \neq \delta$. Since the conjugate $\mathcal{R}^*(\boldsymbol{\eta}) = \frac{1}{2}\|\boldsymbol{\eta}\|^2/\delta$ and $\boldsymbol{\eta}_* = \nabla\mathcal{R}(\mathbf{w}_*) = \delta\mathbf{w}_*$, we get $\mathcal{B}_{\mathcal{GR}}(\mathbf{w}\|\mathbf{w}_*) = \frac{1}{2}(\gamma\|\mathbf{w}\|^2 + \delta\|\mathbf{w}_*\|^2 - 2\delta\mathbf{w}^\top\mathbf{w}_*)$. When $\gamma = \delta$, then this reduces to the divergence used in (7), but otherwise it enables us to reconstruct the gradient of the past objective with respect to the new prior. We define the following K-prior where the weight-divergence is replaced by (11), and use it to obtain $\hat{\mathbf{w}}_{\mathcal{G}}$,

$$\mathcal{K}(\mathbf{w}) = \sum_{i\in\mathcal{D}} \ell\left(h(f^i_{\mathbf{w}_*}), h(f^i_{\mathbf{w}})\right) + \mathcal{B}_{\mathcal{GR}}(\mathbf{w}\|\mathbf{w}_*), \qquad \hat{\mathbf{w}}_{\mathcal{G}} = \arg\min_{\mathbf{w}\in\mathcal{W}} \mathcal{K}(\mathbf{w}) \tag{12}$$

The following theorem states the exact recovery of the retraining-from-scratch solution.

Theorem 2. *For GLMs with strictly convex $\mathcal{R}(\mathbf{w})$ and $\mathcal{G}(\mathbf{w})$, we have $\mathbf{w}_{\mathcal{G}} = \hat{\mathbf{w}}_{\mathcal{G}}$.*

The proof is very similar to (9), where $\delta(\mathbf{w} - \mathbf{w}_*)$ is replaced by $\nabla\mathcal{G}(\mathbf{w}) - \nabla\mathcal{R}(\mathbf{w}_*)$. More examples involving the other adaptation tasks are discussed in Appendix A.

K-priors with limited memory. It is clear that when $\mathcal{M} = \mathcal{D}$, K-priors can recover the solution obtained by retraining from scratch. In practice, using all the past data might be as slow as full retraining, but for small changes we may not always need to use all the past data. The intuition is that, for small changes, the outputs may only change for a handful of data examples (see Fig. 1 for an illustration). A reasonably accurate update can be obtained by simply tweaking just those outputs.

Which inputs should we include? Ideally, we should choose the inputs whose removal incurs the most error in the gradient (9). The error is shown below where it is clear, from the first equality, that the inputs where the predictions disagree the most should be included in $\mathcal{M}$,

$$\nabla\bar{\ell}(\mathbf{w}) - \nabla\mathcal{K}(\mathbf{w}) = \sum_{i\in\mathcal{D}\setminus\mathcal{M}} \boldsymbol{\phi}_i \left[h(f^i_{\mathbf{w}}) - h(f^i_{\mathbf{w}_*})\right] \approx \Big[\sum_{i\in\mathcal{D}\setminus\mathcal{M}} \boldsymbol{\phi}_i h'(f^i_{\mathbf{w}_*})\boldsymbol{\phi}_i^\top\Big](\mathbf{w} - \mathbf{w}_*) \quad (13)$$

Since the discrepencies are unknown beforehand, we can use the approximation shown above, which is obtained by using a first-order expansion: $h(f^i_{\mathbf{w}}) - h(f^i_{\mathbf{w}_*}) \approx h'(f^i_{\mathbf{w}_*})(\nabla f^i_{\mathbf{w}_*})^\top(\mathbf{w} - \mathbf{w}_*)$, where $h'(f^i)$ denotes the derivative. The approximation is conveniently expressed in terms of the *Generalized Gauss-Newton* (GGN) matrix $\mathbf{G}(\mathbf{w}_*) = \sum_{i\in\mathcal{D}} \boldsymbol{\phi}_i h'(f^i_{\mathbf{w}_*})\boldsymbol{\phi}_i^\top$ [32], which can be computed beforehand, irrespective of the adaptation task. The approximation suggests that $\mathcal{M}$ should be chosen to keep the *leftover* GGN matrix orthogonal to $\mathbf{w} - \mathbf{w}_*$. Since $\mathbf{w}$ is unknown beforehand, a reasonable approximation is to choose examples which keep the top-eigenvalues of the GGN matrix, which coincides with the best low-rank approximation of the GGN matrices. Due to this connection, we can use low-rank approximation methods, e.g., choosing examples with high *variance/leverage* $\boldsymbol{\phi}_i^\top \mathbf{G}(\mathbf{w}_*)^{-1}\boldsymbol{\phi}_i$, a popular concept in statistics and kernel methods [13, 1, 12, 7]. This might be computationally difficult, and we will follow [37] to choose $\mathcal{M}$ to be the M examples with the highest $h'(f^i_{\mathbf{w}_*})$ value, which works well for classification problems. The set is referred to as the *memorable past examples* in [37].

A principle for constructing K-priors. The examples above demonstrate the main principle behind the construction of K-priors: construct it such that the gradients can faithfully be reconstructed. This is often possible by exploiting the structure of the learning problem. For example, to replace an old objective of the form (1), with loss $\ell_i^{\text{old}}(f)$ and regularizer $\mathcal{R}^{\text{old}}(\mathbf{w})$, by a new loss $\ell_i^{\text{new}}(f)$ and regularizer $\mathcal{R}^{\text{new}}(\mathbf{w})$, the divergences should be chosen such that they have the following gradients,

$$\nabla\mathbb{D}_w(\mathbf{w}\|\mathbf{w}_*) = \nabla\mathcal{R}^{\text{new}}(\mathbf{w}) - \nabla\mathcal{R}^{\text{old}}(\mathbf{w}), \quad \nabla\mathbb{D}_f(\mathbf{f}(\mathbf{w})\|\mathbf{f}(\mathbf{w}_*)) = \nabla\mathbf{f}(\mathbf{w})^\top\mathbf{B}\mathbf{d}_m$$

where $\mathbf{d}_m$ is an M-length vector with the discrepancy $\nabla\ell_i^{\text{new}}(f^i_{\mathbf{w}}) - \nabla\ell_i^{\text{old}}(f^i_{\mathbf{w}_*})$ as the i'th entry. The matrix $\mathbf{B}$ is added to counter the mismatch between $\mathcal{D}$ and $\mathcal{M}$. Similar construction can be used for other learning objectives. For non-differentiable functions, a Bayesian version can be used with the Kullback-Leibler (KL) divergence (we discuss this in the next section). This is easy to do with exponential family distributions, which have a one-to-one mapping with Bregman divergences [4].

With this general principle, our priors automatically satisfy many principles of a 'good' prior-construction, as suggested in [46]; see Principles D1, D2, D3, D6, and D7 in the paper. To summarize those, our prior assigns low complexity values to the base model ($\mathcal{K}(\mathbf{w}_*) = 0$), it is informative, it measures what the model does rather than its numerical values, it exploits the structure of the objective, it provides means to adapt to the changes, and it is computationally feasible.

4 K-priors: Extensions and Connections

We discuss extensions to Deep Learning and Bayesian Learning, and connections to many others.

Weight-Priors. Gaussian weight-priors [10, 24] can be seen as K-priors with restrictive approximations, e.g., priors using the Hessian/GGN matrix: $(\mathbf{w} - \mathbf{w}_*)^\top(\mathbf{G}(\mathbf{w}_*) + \delta\mathbf{I})(\mathbf{w} - \mathbf{w}_*)$ approximate the K-prior given in (7). We can show this by using a first-order expansion in (8) (similar to the one used in (13)) to see that the K-prior gradient is the same as the weight-prior with the GGN (=Hessian),

$$\nabla\mathcal{K}(\mathbf{w}) \approx \sum_{i\in\mathcal{D}} \boldsymbol{\phi}_i \left[h'(\boldsymbol{\phi}_i^\top\mathbf{w}_*)\boldsymbol{\phi}_i^\top(\mathbf{w} - \mathbf{w}_*)\right] + \delta(\mathbf{w} - \mathbf{w}_*) \;\approx\; (\mathbf{G}(\mathbf{w}_*) + \delta\mathbf{I})(\mathbf{w} - \mathbf{w}_*).$$

Weight-priors approximate the functional divergence term by a Hessian/GGN matrix evaluated at $\mathbf{w}_*$. This takes $P \times P$ memory. In contrast, K-priors use past memory with the gradient evaluated at the *new* $\mathbf{w}$. This could be more accurate, but comes at the cost of storing enough examples. Since the rank of the feature matrix is at most equal to $\min(P, N)$ where N is the data size, we expect the memory requirements for K-priors to be not much bigger than weight-priors. In addition, we can always apply sketching or singular-value decomposition to minimize the memory requirements.

K-priors for Deep Learning and Connections to Knowledge Distillation. We now discuss the application to deep learning. It is clear that the functional term in K-priors is similar to Knowledge distillation (KD) [21], which is a popular approach for model compression in classification problems with the Softmax link function (the following expression is taken from [31]),

$$\ell_{\text{KD}}(\mathbf{w}) = \lambda \sum_{i \in \mathcal{D}} \ell\left(y_i, h(f^i_{\mathbf{w}})\right) + (1-\lambda) \sum_{i \in \mathcal{D}} \ell\left(h(f^i_{\mathbf{w}_*}/T),\, h(f^i_{\mathbf{w}})\right), \tag{14}$$

The base model predictions are often scaled with a temperature parameter $T > 0$ and $\lambda \in [0, 1]$. KD can be seen as a special case of K-priors without the weight-space term ($\tau = 0$), and an assymetric Bregman divergence for the function-space term, due to the temperature parameter. K-priors extend KD in three ways by (i) adding the weight-space term, (ii) allowing general link functions or divergence functions, and (iii) using a small number of examples in $\mathcal{M}$ instead of the whole dataset. With these extensions, K-priors can handle adaptation tasks other than compression. Due to their similarity, it is also possible to borrow tricks used in KD to improve the performance of K-priors.

KD often yields solutions that are better than retraining from scratch. Theoretically the reasons behind this are not understood well, but we now show that K-priors and KD have a common property. Unlike GLMs, K-priors for deep learning do not exactly reconstruct the gradient of the past training objective, and there is an additional left-over term (derivation is in Appendix B),

$$\nabla \mathcal{K}(\mathbf{w}) = \underbrace{\sum_{i \in \mathcal{D}} \nabla f^i_{\mathbf{w}} \left[h(f^i_{\mathbf{w}}) - y_i\right] + \delta \mathbf{w}}_{=\nabla \bar{\ell}(\mathbf{w})} - \underbrace{\sum_{i \in \mathcal{D}} \nabla f^i_{\mathbf{w}} r^i_{\mathbf{w}_*} - \delta \mathbf{w}_*}_{\text{Additional term since } \nabla f^i_{\mathbf{w}} \neq \nabla f^i_{\mathbf{w}_*}}, \tag{15}$$

where $r^i_{\mathbf{w}_*} := h(f^i_{\mathbf{w}_*}) - y_i$ is the residual obtained by $\mathbf{w}_*$. Remarkably, the gradient of the KD objective has this exact same term (derived by taking the gradient of (14), with $\delta = 0$, $T = 1$),

$$\nabla \ell_{\text{KD}}(\mathbf{w}) = \sum_{i \in \mathcal{D}} \nabla f^i_{\mathbf{w}} \left[h(f^i_{\mathbf{w}}) - y_i\right] - (1-\lambda) \sum_{i \in \mathcal{D}} \nabla f^i_{\mathbf{w}} r^i_{\mathbf{w}_*}.$$

The additional term adds large gradients to push away from the high residual examples (the examples the teacher did not fit well). It is a desirable property to trust the knowledge of the base model and use it to improve the adapted model. It is possible that this term helps the performance of KD too, but more investigations are required to show this. We see in our empirical results that this term does not harm the performance and applications to deep learning at small scale show similar trends to GLMs.

Adding/removing data for SVMs. K-prior regularized training yields equivalent solutions to the adaptation strategies used in SVM to add/remove data examples. K-priors can be shown to be equivalent to the primal formulation of such strategies [29]. The key trick to show the equivalence is to use the representer theorem which we will now illustrate for 'Add Data' task in (10). Let $\mathbf{\Phi}_+$ be the $(N+1) \times P$ feature matrix obtained on the dataset $\mathcal{D} \cup j$, then by the representer theorem we know that there exists a $\boldsymbol{\beta} \in \mathbb{R}^{N+1}$ such that $\mathbf{w}_+ = \mathbf{\Phi}_+^\top \boldsymbol{\beta}$. Taking the gradient of (10), and multiplying by $\mathbf{\Phi}_+$, we can write the optimality condition as,

$$0 = \mathbf{\Phi}_+^\top \nabla[\ell_j(\mathbf{w}_+) + \mathcal{K}(\mathbf{w}_+)] = \sum_{i \in \mathcal{D} \cup j} \left(\nabla_f \ell(y_i, h(f))|_{f = \boldsymbol{\beta}_i^\top \mathbf{k}_{i,+}} \right) \mathbf{k}_{i,+} + \delta \mathbf{K}_+ \boldsymbol{\beta} \tag{16}$$

where $\mathbf{K}_+ = \mathbf{\Phi}_+ \mathbf{\Phi}_+^\top$ and its i'th column is denoted by $\mathbf{k}_{i,+}$. This is exactly the gradient of the primal objective in the function-space defined over the full batch $\mathcal{D} \cup j$; see Equation 3.6 in Chapelle [11]. The primal strategy is equivalent to the more common dual formulations [9, 50, 22, 17, 40, 25, 49]. The function-space formulations could be computationally expensive, but speed-ups can be obtained by using support vectors. This is similar to the idea of using limited memory in K-priors in Section 3.

K-priors for Bayesian Learning and Connections to GPs. K-priors can be seamlessly used for adaptation within a Bayesian learning framework. Consider a Gaussian approximation $q_*(\mathbf{w})$ trained

on a variational counterpart of (1) with prior $p(w) \propto \exp\left[-\mathcal{R}(\mathbf{w})\right]$, and its adapted version where we add data, shown below,

$$q_*(\mathbf{w}) = \arg\min_{q \in Q} \sum_{i \in \mathcal{D}} \mathbb{E}_q[\bar{\ell}_i(\mathbf{w})] + \mathbb{D}_{KL}[p \,\|\, q], \quad q_+(\mathbf{w}) = \arg\min_{q \in Q} \sum_{i \in \mathcal{D} \cup j} \mathbb{E}_q[\ell_i(\mathbf{w})] + \mathbb{D}_{KL}[p \,\|\, q].$$

Assuming the same setup as Section 3, we can recover $q_+(\mathbf{w})$ by using $q_{\mathcal{K}}(\mathbf{w}) \propto \exp\left[-\mathcal{K}(\mathbf{w})\right]$ where we use the K-prior defined in (7) (note that normalization constant of $q_{\mathcal{K}}$ is not required),

$$\hat{q}_+(\mathbf{w}) = \arg\min_{q \in Q} \; \mathbb{E}_q[\ell_j(\mathbf{w})] + \mathbb{D}_{KL}[q \,\|\, q_{\mathcal{K}}], \tag{17}$$

A proof easily follows using (9) and is given in Appendix C. In fact, when this Bayesian extension is written in the function-space similarly to (16), it is related to the online updates used in GPs [14]. When $q_{\mathcal{K}}$ is built with limited memory, as described in (3), the application is similar to sparse variational GPs, but now data examples are used as inducing inputs. These connections are discussed in more detail in Appendix C. Our K-prior formulation in Section 3 operates in the weight-space and can be easily trained with first-order methods, however an equivalent formulation in the function space can also be employed, as is clear from these connections. The above extensions can be extended to handle arbitrary exponential-family approximations by appropriately defining K-priors using KL divergences. We omit these details since this topic is more suitable for a Bayesian version of this paper. These extensions will be published in a separate study.

Memory-Based Methods for Deep Continual Learning. K-priors are also closely related to recent functional regularization approaches proposed for deep continual learning [28, 38, 5, 48, 37, 6]. The approach of [45] uses a mix of function and weight-space divergences (see Eq. 2 in their paper). The recent approach of [37] is perhaps the closest to ours which uses discrepencies $h(f^i_{\mathbf{w}}) - h(f^i_{\mathbf{w}_*})$ at selected inputs, weighted by a kernel matrix. They also propose to select the inputs according to $h'(f^i_{\mathbf{w}_*})$, which is consistent with our theory (see (13)). The gradient episodic memory (GEM) approach [30] is also related, where the goal is to ensure that the $\sum_{i \in \mathcal{M}}[\ell(y_i, f^i_{\mathbf{w}_*}) - \ell(y_i, f^i_{\mathbf{w}})] < 0$. This is similar to the student-teacher transfer used in SVM where the loss of the student is regularized using the model output of the teacher (see Eq. 7 in [51] for an example). [30] relax the optimization problem to write it in terms of the gradients. This is similar to K-priors, except K-priors rely on gradients within a first-order method, which is easier to train than the dual approach used in [30].

Most of these approaches do not employ the weight-space divergence, and sometimes even the function-space divergence is replaced by the Euclidean one [5, 6]. Often, the input locations are sampled randomly, or using a simple replay method [6] which could be suboptimal. The approach in [48] uses inducing inputs which is closely connected to the online GP update. The method we proposed does not contradict with these, but gives a more direct way to choose the points where the gradient errors are taken into consideration.

5 Experimental Results

We compare the performance of K-priors to retraining with full-batch data ('Batch') and a retraining with replay from a small memory ('Replay'). To be fair, we use the same memory as the one chosen for K-priors (by choosing points with highest $h'(f^i_{\mathbf{w}_*})$ as detailed in Section 3). The main difference between Replay and K-priors is that the former uses the true label while K-priors use model-predictions. Memory chosen randomly often gives much worse results. We compare these three methods on the four adaptation tasks: 'Add Data', 'Remove Data', 'Change Regularizer', and 'Change architecture'. For the 'Add Data' task, we also compare to Weight-Priors with GGN.

Our overall finding is that, for GLMs and deep learning on small problems, K-priors can achieve the same performance as Batch, but with a small fraction of data (often 2-10% of the full data (Figure 1, right, and Figure 2). Replay does much worse for small memory size, which clearly shows the advantage of using the model predictions in K-priors (see the discussion after Theorem 1). Weight priors generally perform well, but they can do very badly when the adaptation requires a change in the memorable examples (see Figure 3). Finally, for large deep-learning problems, results are promising but more investigations are required with extensive hyperparameter tuning. Several additional experiments are in Appendix D. We now give details of each experiment.

Logistic Regression on the 'UCI Adult' dataset. This is a binary classification problem consisting of 16,100 examples to predict income of individuals. We randomly sample 10% of the training

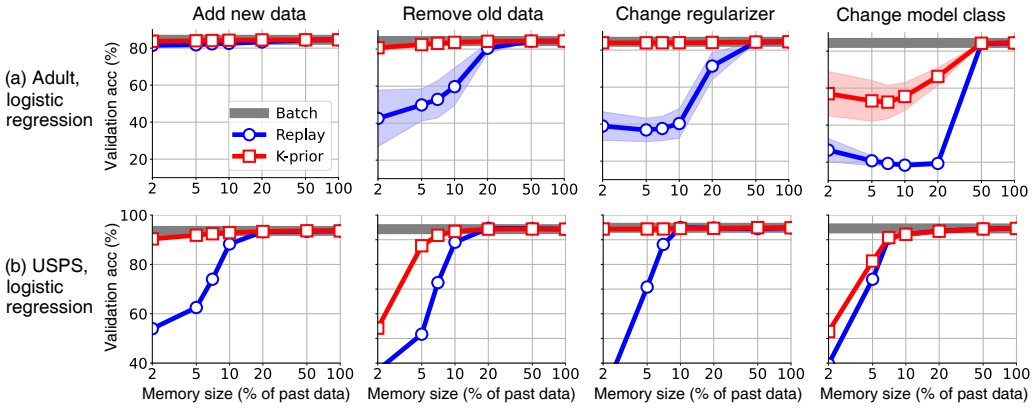

Figure 2: K-priors (red squares) match Batch (grey) while mostly using 2-5% of the data (for only 3 tasks a larger fraction is required). K-priors always outperforms Replay which uses the true labels. K-priors replace the labels by the model predictions (see the discussion after Theorem 1).

data (1610 examples), and report mean and standard deviation over 10 such splits. For training, we use the L-BFGS optimizer for logistic regression with polynomial basis. Results are summarized in Figure 2(a). For the 'Add Data' task, the base model uses 9% of the data and we add 1% new data. For 'Remove Data', we remove 100 data examples (6% of the training set) picked by sorting $h'(f^i_{\mathbf{w}_*})$). For the 'Change Regularizer' task, we change the L_2 regularizer from $\delta = 50$ to 5, and for 'Change Model Class', we reduce the polynomial degree from 2 to 1. K-priors perform very well on the first three tasks, remaining very close to Batch, even when the memory sizes are down to 2%. 'Changing Model Class' is slightly challenging, but K-priors still significantly out-perform Replay.

Logistic Regression on the 'USPS odd vs even' dataset. The USPS dataset consists of 10 classes (one for each digit), and has 7,291 training images of size 16×16. We split the digits into two classes: odd and even digits. Results are in Figure 2(b). For the 'Add Data' task, we add all examples for the digit 9 to the rest of the dataset, and for 'Remove Data' we remove the digit 8 from the whole dataset. By adding/removing an entire digit, we enforce an *inhomogeneous* data split, making the tasks more challenging. The 'Change Regularizer' and 'Change Model Class' tasks are the same as the Adult dataset. K-priors perform very well on the 'Add Data' and 'Change Regularizer' tasks, always achieving close to Batch performance. For 'Remove Data', which is a challenging task due to inhomogenity, K-priors still only need to store 5% of past data to maintain close to 90% accuracy, whereas Replay requires 10% of the past data.

Neural Networks on the 'USPS odd vs even' dataset. This is a repeat of the previous experiment but with a deep neural network (a 1-hidden-layer MLP with 100 units). Results are in Figure 1 (right). The 'Change Regularizer' task now changes $\delta = 5$ to 10, and the 'Change Architecture' task compresses the architecture from a 2-hidden-layer MLP (100 units per layer) to a 1-hidden-layer MLP with 100 units. We see that even with neural networks, K-priors perform very well, similarly out-performing Replay and remaining close to the Batch solution at small memory sizes.

Weight-priors vs K-priors. As discussed in the main text, weight-priors can be seen as an approximation of K-priors where $h'(f^i_{\mathbf{w}})$ are replaced by 'stale' $h'(f^i_{\mathbf{w}_*})$, evaluated at the old $\mathbf{w}_*$. In Figure 3(a), we visualize these 'stale' $h'(f^i_{\mathbf{w}_*})$ and compare them to K-priors which obtains values close to the ones found by Batch. Essentially, for the points at the diagonal the match is perfect, and we see that it is the case for K-priors but not for the weight-priors. We use logistic regression on the USPS data (the 'Add Data' task). This inhomogeneous data split is difficult for weight-priors, and we show in Figure 3(b) that weight-priors do not perform well. For homogeneous data splits, weight-priors do better than this, but this result goes to show that they do not have any mechanisms to fix the mistakes made in the past. In K-priors, we can always change $\mathcal{M}$ to improve the performance. We provide more plots and results in Appendix D, including results for weight-priors on all the 'Add data' tasks we considered in this paper.

MNIST and CIFAR-10, neural networks. Finally, we discuss results on larger problems in deep learning. We show many adaptation tasks in Appendix D for 10-way classification on MNIST

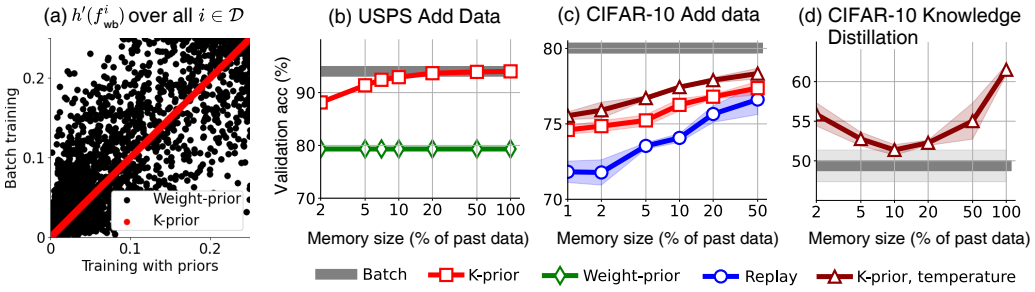

Figure 3: (a) When compared at the Batch solution for the 'Add Data' task on USPS, weight priors give incorrect values of $h'(f^i_{\mathbf{w}})$ (shown with black dots, each dot correspond to a data examples). Points on the diagonal means a perfect match which is the case for K-priors (show with red dots). (b) Due to this, weight-priors (green diamonds) perform worse than K-priors (red squares). (c) For the 'Add data' task on CIFAR-10, K-priors outperform replay (blue circles), but performance can still be improved by using a temperature parameter (dark-red triangles). (d) The same is true for knowledge distillation [21], and we see that we can reduce memory size while still performing better than the student model. The dip in the middle, we believe, is due to suboptimal hyperparameter tuning.

[26] with MLPs and 10-way classification on CIFAR-10 with CifarNet [53], trained with the Adam optimizer [23]. In Figure 3(c) we show one representative result for the 'Add data' task with CIFAR-10, where we add a random 10% of CIFAR-10 training data to the other 90% (mean and standard deviation over 3 runs). Although vanilla K-priors outperform Replay, there is now a bigger gap between K-prior and Batch even with 50% past data stored. However, when we use a temperature (similar to knowledge distillation in (14) but with the weight term included), K-priors improves.

A similar result is shown in Figure 3(d) for knowledge distillation ($\delta = 0$ but with a temperature parameter) where we are distill from a CifarNet teacher to a LeNet5-style student (details in Appendix D). Here, K-priors with 100% data is equivalent to Knowledge Distillation, but when we reduce the memory size using our method, we still outperform Batch (which is trained from scratch on all data). There is a dip in performance at memory sizes of 10% past data, which we believe is due to the use of a suboptimal τ or λ. More empirical effort is require to tune various hyperparameters to get a consistent behavior at all memory sizes. Overall, our initial effort here suggests that K-priors can do better than Replay, and have potential to give better results with more hyperparameter tuning.

6 Discussion

In this paper, we proposed a class of new priors, called K-prior. We show general principles of obtaining accurate adaptation with K-priors which are based on accurate gradient reconstructions. These principles have many properties that a good prior is expected to have [46]. The prior applies to a wide-variety of adaptation tasks for a range of models, and helps us to connect many existing, seemingly-unrelated adaptation strategies in ML. Based on our adaptation principles, we derived practical methods to enable adaptation by tweaking models' predictions at a few past examples. This is analogous to adaptation in humans and animals where past experiences is used for new situations. In practice, the amount of required past memory seems sufficiently low for many tasks. Overall, K-priors provide an intuitive yet practical mechanisms for generic adaptation in ML.

The financial and environmental costs of retraining are a huge concern for ML practitioners, which can be reduced with quick adaptations. During this work, we realized how little work has been done on this topic. The current pipelines and designs are specialized for an offline, static setting. Our approach here pushes towards a simpler design which will support a more dynamic setting. The approach can eventually lead to new systems that learn quickly and flexibly, and also act sensibly across a wide range of tasks. This opens a path towards systems that learn incrementally in a continual fashion, with the potential to fundamentally change the way ML is used in scientific and industrial applications. We hope that this work will help others to do more work towards this goal in the future. We ourselves will continue to push this work in that direction.

References

[1] Ahmed El Alaoui and Michael W Mahoney. Fast randomized kernel methods with statistical guarantees. *arXiv preprint arXiv:1411.0306*, 2014.

[2] Shuang Ao, Xiang Li, and Charles Ling. Fast generalized distillation for semi-supervised domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[3] Jordan Ash and Ryan P Adams. On warm-starting neural network training. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3884–3894. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/288cd2567953f06e460a33951f55daaf-Paper.pdf.

[4] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *Journal of machine learning research*, 6(Oct):1705–1749, 2005.

[5] Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

[6] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

[7] Daniele Calandriello, Luigi Carratino, Alessandro Lazaric, Michal Valko, and Lorenzo Rosasco. Gaussian process optimization with adaptive sketching: Scalable and no regret. In *Conference on Learning Theory*, pages 533–557. PMLR, 2019.

[8] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, page 463–480, May 2015. doi: 10.1109/SP.2015.35.

[9] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, pages 409–415, 2001.

[10] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[11] Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5): 1155–1178, May 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.5.1155.

[12] Michael B Cohen, Cameron Musco, and Christopher Musco. Ridge leverage scores for low-rank approximation. *arXiv preprint arXiv:1511.07263*, 6, 2015.

[13] R Dennis Cook. Detection of influential observation in linear regression. *Technometrics*, 19(1): 15–18, 1977.

[14] Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural computation*, 14 (3):641–668, 2002.

[15] Dennis DeCoste and Kiri Wagstaff. Alpha seeding for support vector machines. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, page 345–349. ACM Press, 2000. ISBN 978-1-58113-233-5. doi: 10.1145/347090.347165. URL http://portal.acm.org/citation.cfm?doid=347090.347165.

[16] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice. *arXiv preprint arXiv:1903.05202*, 2019.

[17] Hua Duan, Hua Li, Guoping He, and Qingtian Zeng. Decremental learning algorithms for nonlinear langrangian and least squares support vector machines. In *Proceedings of the First International Symposium on Optimization and Systems Biology (OSB'07)*, pages 358–366. Citeseer, 2007.

[18] Antonio Ginart, Melody Y. Guan, Gregory Valiant, and James Zou. Making ai forget you: Data deletion in machine learning. *arXiv:1907.05012 [cs, stat]*, Nov 2019. URL http://arxiv.org/abs/1907.05012. arXiv: 1907.05012.

[19] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.

[20] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030*, 2019.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[22] Masayuki Karasuyama and Ichiro Takeuchi. Multiple incremental decremental learning of support vector machines. *IEEE Transactions on Neural Networks*, 21(7):1048–1059, Jul 2010. ISSN 1045-9227, 1941-0093. doi: 10.1109/TNN.2010.2048039.

[23] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[24] 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. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[25] Pavel Laskov, Christian Gehl, Stefan Krüger, Klaus-Robert Müller, Kristin P Bennett, and Emilio Parrado-Hernández. Incremental support vector learning: Analysis, implementation and applications. *Journal of machine learning research*, 7(9), 2006.

[26] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[27] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. *arXiv preprint arXiv:1703.08475*, 2017.

[28] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[29] Zhizheng Liang and YouFu Li. Incremental support vector machine learning in the primal and applications. *Neurocomputing*, 72(10):2249–2258, Jun 2009. ISSN 0925-2312. doi: 10.1016/j.neucom.2009.01.001.

[30] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*, 2017.

[31] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. *arXiv preprint arXiv:1511.03643*, 2015.

[32] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[33] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[34] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Variational bayesian unlearning. *arXiv:2010.12883 [cs, stat]*, Oct 2020. URL http://arxiv.org/abs/2010.12883. arXiv: 2010.12883.

[35] Frank Nielsen. On a variational definition for the jensen-shannon symmetrization of distances based on the information radius. *Entropy*, 23(4):464, 2021.

[36] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. Challenges in deploying machine learning: a survey of case studies. *arXiv:2011.09926 [cs]*, Jan 2021. URL http://arxiv.org/abs/2011.09926. arXiv: 2011.09926.

[37] Pingbo Pan, Siddharth Swaroop, Alexander Immer, Runa Eschenhagen, Richard E Turner, and Mohammad Emtiyaz Khan. Continual deep learning by functional regularisation of memorable past. *arXiv preprint arXiv:2004.14070*, 2020.

[38] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[39] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810*, 2018.

[40] Enrique Romero, Ignacio Barrio, and Lluís Belanche. Incremental and decremental learning for linear support vector machines. In *International Conference on Artificial Neural Networks*, pages 209–218. Springer, 2007.

[41] Sebastian Ruder, Parsa Ghaffari, and John G Breslin. Knowledge adaptation: Teaching to adapt. *arXiv preprint arXiv:1702.02052*, 2017.

[42] Nikolaos Sarafianos, Michalis Vrigkas, and Ioannis A Kakadiaris. Adaptive svm+: Learning with privileged information for domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2637–2644, 2017.

[43] Simo Särkkä, Arno Solin, and Jouni Hartikainen. Spatiotemporal learning via infinite-dimensional bayesian filtering and smoothing: A look at gaussian process regression through kalman filtering. *IEEE Signal Processing Magazine*, 30(4):51–61, 2013.

[44] Sebastian Schelter. amnesia–towards machine learning models that can forget user data very fast. In *1st International Workshop on Applied AI for Database Systems and Applications (AIDB19)*, 2019.

[45] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

[46] Daniel Simpson, Håvard Rue, Andrea Riebler, Thiago G. Martins, and Sigrunn H. Sørbye. Penalising model component complexity: A principled, practical approach to constructing priors. *Statistical Science*, 32(1):1–28, 2017. ISSN 08834237, 21688745. URL http://www.jstor.org/stable/26408114.

[47] Arno Solin, James Hensman, and Richard E Turner. Infinite-horizon gaussian processes. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/b865367fc4c0845c0682bd466e6ebf4c-Paper.pdf.

[48] Michalis K. Titsias, Jonathan Schwarz, Alexander G. de G. Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with gaussian processes, 2020.

[49] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 343–352. ACM, Aug 2014. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623661. URL https://dl.acm.org/doi/10.1145/2623330.2623661.

[50] Amund Tveit, Magnus Lie Hetland, and Håavard Engum. Incremental and decremental proximal support vector classification using decay coefficients. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 422–429. Springer, 2003.

[51] Vladimir Vapnik and Rauf Izmailov. Learning using privileged information: similarity control and knowledge transfer. *J. Mach. Learn. Res.*, 16(1):2023–2049, 2015.

[52] Zeyi Wen, Bin Li, Ramamohanarao Kotagiri, Jian Chen, Yawen Chen, and Rui Zhang. Improving efficiency of svm k-fold cross-validation by alpha seeding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[53] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph of the results section. Also see line 193 where we discuss the relations with weight priors.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We generally believe that this has positive impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Theorem 1 and 2.
   (b) Did you include complete proofs of all theoretical results? [Yes] These are in the Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We add details in the supplementary material, but only those details which do not reveal our identities.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5. We also add details in Appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Fig. 2 and 3.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A] We did not use such huge resources in this paper.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]