Proximal Mapping for Deep Regularization

Mao Li, Yingyi Ma, Xinhua Zhang
Department of Computer Science, University of Illinois at Chicago
Chicago, IL 60607
{mli206, yma36, zhangx}@uic.edu

Abstract

Underpinning the success of deep learning is effective regularizations that allow a variety of priors in data to be modeled. For example, robustness to adversarial perturbations, and correlations between multiple modalities. However, most regularizers are specified in terms of hidden layer outputs, which are not themselves optimization variables. In contrast to prevalent methods that optimize them indirectly through model weights, we propose inserting proximal mapping as a new layer to the deep network, which directly and explicitly produces well regularized hidden layer outputs. The resulting technique is shown well connected to kernel warping and dropout, and novel algorithms were developed for robust temporal learning and multiview modeling, both outperforming state-of-the-art methods.

1 Introduction

The success of deep learning relies on massive neural networks that often considerably out-scale the training dataset, defying the conventional learning theory [1, 2]. Regularization has been shown essential and a variety of forms are available. For example, invariances to transformations such as rotation [3] have been extended beyond group-based diffeomorphisms to indecipherable transformations that are only exemplified by pairs of views [4], e.g., sentences uttered by the same person. Prior regularities are also commonly available **a)** *within* layers of neural networks, such as sparsity [5], spatial invariance in convolutional nets, structured gradient that accounts for data covariance [6]; **b)** *between* layers of representation, such as stability under dropout and adversarial perturbations of preceding layers [7], contractivity between layers [8], and correlations in hidden layers among multiple views [9, 10]; and **c)** at batch level, e.g., disentangled representation and multiple modalities.

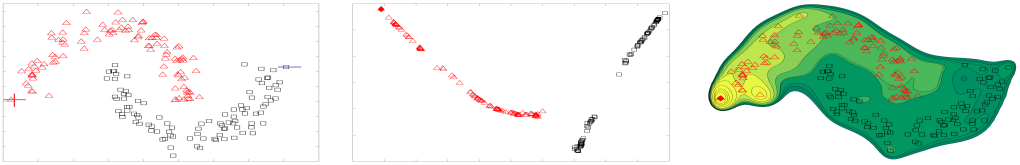
The most prevalent approach to incorporating priors is regularization, which leads to the standard regularized risk minimization (RRM) for a given dataset $\mathcal{D}$, empirical distribution $\tilde{p}$, and loss ℓ:

$$\min_f \ \mathbb{E}_{x\sim\tilde{p}}[\ell(f(x))] + \Gamma(f) + \sum\nolimits_i \Omega_i(\{h_i(x,f)\}_{x\in\mathcal{D}}). \tag{1}$$

Here f is the predictor (e.g., neural network), and Γ is the data-independent regularizer (e.g., L_2 norm), and Ω_i is the data-dependent regularizer on the i-th layer output h_i under f (e.g., invariance of h_i with respect to the i-th step input x_i in an RNN). Note Ω_i can involve multiple layers (e.g., contractivity), or be decomposed over training examples. Optimization techniques such as end-to-end training have produced strong performance, along with progresses in the global analysis of the solution [e.g., 11]. However, all these analyses make assumptions on the landscape of the objective function, which, although often satisfied by the empirical risk $\mathbb{E}_{x\sim\tilde{p}}[\ell(f(x))]$, are typically violated or complicated by the addition of data-dependant regularizers Ω_i. The nontrivial contention between accurate prediction and faithful regularization can often confound the optimization of model weights.

A natural question therefore arises: is it possible to further improve the effectiveness of regularization, potentially not only through the development of new solvers and analysis for RRM, but also through novel mechanisms of incorporating regularization? Although the former approach has been studied intensively, we hypothesize and will demonstrate empirically that the latter approach can

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.



(a) Original two-moon dataset (b) Representation after prox-map (c) Contour to ♦ after prox-map

Figure 1: (a) The two-moon dataset with only two labeled examples '+' and '−' (left and right), but abundant unlabeled examples that reveal the inherent structure; (b) Representation inferred from top-2 kernel PCA based on the proximal mapping with gradient flatness and Gaussian kernel (see §3); (c) contour of distance to the leftmost point ♦, based on the result of proximal mapping.

be surprisingly effective. Our key intuition is, now that Ω_i is specified in terms of the hidden layer output h_i (which is determined by f), can we directly optimize h_i as opposed to indirectly through f? Treating h_i as ground variables and optimizing them jointly with model weights has been used by [12]. However, their motivation is on accelerating the optimization rather than improving the model.

It turns out that this idea can be conveniently implemented by leveraging the tool of proximal mapping (hence the name ProxNet), which has been extensively used in optimization to enforce structured solutions such as sparsity [13]. Given a closed convex set $C \subseteq \mathbb{R}^n$ and a convex function $R : \mathbb{R}^n \to \mathbb{R}$ which favor certain desirable prior (e.g., ℓ_1 norm), the proximal mapping $\mathsf{P}_R : \mathbb{R}^n \to \mathbb{R}^n$ is defined as

$$\mathsf{P}_R(x) := \arg\min_{z \in C} \{R(z) + \tfrac{\lambda}{2} \|z - x\|^2\}, \quad \text{where the norm is } L_2. \tag{2}$$

In essence, R and C encourage the mapping to respect the prior encoded by R, while remaining in the vicinity of x. For example, Figure 1a shows the two-moon dataset with only two labeled examples and many unlabeled ones. Figures 1b and 1c show the resulting representation and warped distance where R accounts for the underlying manifold, making the classification trivial (§3).

In a deep network, the proximal mapping can be inserted after any layer to turn h_i into $\mathsf{P}_{\Omega_i}(h_i)$, and backpropagate through it. **Why** does this yield a more effective implementation of regularization? First of all, it provides the modularity of decoupling regularization from supervised learning — the regularization is encapsulated *within* the proximal layer that is *free of weights*, and the resulting $\mathsf{P}_{\Omega_i}(h_i)$ is directly enforced to comply with the prior rather than indirectly through the optimization of weights in f. This frees weight optimization from simultaneously catering to unsupervised structures and supervised performance metrics, which plagues the conventional RRM. Such an advantage will be confirmed in our experiments of end-to-end training that are highly efficient (§5.1).

Secondly, proximal mapping can be interpreted as an intermediate step of denoising, where $\mathsf{P}_{\Omega_i}(h_i)$ is a *cleaned* version of h_i that conforms to the prior. This ensures that the downstream layers are presented with well regularized inputs, which will presumably facilitate their own learning. By gradually increasing λ, such a manual morphing can be annealed, allowing the upstream layers (e.g., feature extractors) to approach weight values that by themselves produce well-regularized h_i. ProxNet is also readily connected with meta-learning (§B) because of the bi-level optimization setup, where the proximal layer plays a similar role to base-learners.

Finally, $\mathsf{P}_{\Omega_i}(h_i)$ can be carried out on a mini-batch $\mathcal{B}$, where R is defined on a set $\{h_i(x)\}_{x \in \mathcal{B}}$. It also extends flexibly to regularizers that account for multiple layers, e.g., invariance of h_i to h_{i-1}.

This paper will first review the existing building blocks of deep networks through the lens of proximal mapping (§2), and then unravel its non-trivial connections with regularization when the latter is quadratic (e.g., manifold smoothness) or non-quadratic (e.g., dropout). Afterwards, two novel ProxNets will be introduced that achieve robust recurrent modeling (§4) and multiview learning (§5). Extensive experiments show that ProxNet outperforms state-of-the-art prediction models (§6).

Related Work ProxNet instantiates the differentiable optimization framework laid by OptNet [14, 15] along with [16–25], which provides recipes for differentiating through an optimization layer. In contrast, our focus is not on optimization, but on using ProxNet to model the priors in the data, which typically involves an (inner) unsupervised learning task such as CCA. More detailed discussions on the relationship between ProxNet and OptNet or related works are available in Appendix A.

Another proximal-like operator was found in "sparsemap" operations [26–28]. However, they target a different application of incorporating structured sparsity in attention weights for a *single* instance, rather than at a mini-batch level where ProxNet is applied for multiview learning.

2 Proximal Mapping as a Primitive Construct in Deep Networks

Proximal mapping is highly general, encompassing most primitive operations in deep learning [13, 29]. For example, any activation function σ with $\sigma'(x) \in (0, 1]$ (e.g., sigmoid) is indeed a proximal map with $C = \mathbb{R}^n$ and $R(x) = \int \sigma^{-1}(x)\,\mathrm{d}x - \frac{1}{2}x^2$, which is convex. The ReLU and hard tanh activations can be recovered by $R = 0$, with $C = [0, \infty)$ and $C = [-1, 1]$, respectively. Soft-max transfer $\mathbb{R}^n \ni x \mapsto (e^{x_1}, \ldots, e^{x_n})^\top / \sum_i e^{x_i}$ corresponds to $C = \{x \in \mathbb{R}^n_+ : \mathbf{1}^\top x = 1\}$ and $R(x) = \sum_i x_i \log x_i - \frac{1}{2}x_i^2$, which are convex. Batch normalization maps $x \in \mathbb{R}^n$ to $(x - \mu\mathbf{1})/\sigma$, where $\mathbf{1}$ is a vector of all ones, and μ and σ are the mean and standard deviation of the elements in x, respectively. This mapping can be recovered by $R = 0$ and $C = \{x : \|x\| = \sqrt{n}, \mathbf{1}^\top x = 0\}$. Although C is not convex, this $\mathsf{P}_R(x)$ must be a singleton for $x \neq 0$. In general, R and C can be nonconvex making $\mathsf{P}_R(z)$ set-valued, and we only need differentiation at one element [30–32].

Kernelization. Proximal mapping can be trivially extended to reproducing kernel Hilbert spaces (RKHSs), allowing non-vectorial data to be encoded [33] and invariances to be hard wired [34, 35]. Assume an RKHS $\mathcal{H}$ employs a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Given a convex functional $R : \mathcal{H} \to \mathbb{R}$, a proximal map $\mathsf{P}_R : \mathcal{H} \to \mathcal{H}$ can be defined in exactly the same form as (2), with L_2 norm replaced by RKHS norm.

3 Connecting Proximal Mapping to RRM on Shallow Models

We first illustrate the connection between RRM and proximal mapping. To focus on the core idea, we use shallow models with no hidden layer. Letting $k_x := k(x, \cdot)$ be the kernel representer of x and R be the regularizer encoding preference on f, we can write the two formulations as follows:

$$\text{P1: } \min_{f \in \mathcal{H}} \mathbb{E}_{x \sim \tilde{p}}[\ell(\langle f, k_x \rangle_{\mathcal{H}})] + R(f) \quad \text{v.s.} \quad \text{P2: } \min_{h \in \mathcal{H}} \mathbb{E}_{x \sim \tilde{p}}[\ell(\langle h, c_x \rangle_{\mathcal{H}})] + \lambda^2 \|h\|^2_{\mathcal{H}}\,, \tag{3}$$

$$\text{where} \quad c_x := \mathsf{P}_R(k_x) = \arg\min_{g \in \mathcal{H}} \left\{ \tfrac{\lambda}{2} \|g - k_x\|^2_{\mathcal{H}} + R(g) \right\}. \tag{4}$$

i) $R(f)$ is a positive semi-definite (PSD) quadratic. Examples of this simplest case include graph Lapalacian $R_l(f) := \sum_{ij} w_{ij}(f(x_i) - f(x_j))^2$ and gradient penalty $R_g(f) := \sum_i \|\nabla f(x_i)\|^2$. They both enforce smoothness on a data manifold. Since the gradient operator $\nabla R : f \mapsto \nabla R(f)$ is linear, we denote its eigenvalues and eigenfunctions as $\{\mu_i, \phi_i\}$. Further, taking derivative of g in (4), we derive a closed-form for the proximal map as $c_x = \lambda(\lambda I + \nabla R)^{-1} k_x$, where I is the identity operator. The contour in Figure 1c was plotted exactly by using the pairwise distance $\|c_x - c_{x'}\|_{\mathcal{H}}$, based on which the new data representation in Figure 1b was extracted using the top-2 principal components.

To connect P1 and P2, let $h = \lambda^{-1}(\lambda I + \nabla R) f$. Then $\langle f, k_x \rangle_{\mathcal{H}} = \langle h, c_x \rangle_{\mathcal{H}}$ (i.e., same prediction) and

$$R(f) = \tfrac{1}{2} \sum\nolimits_i \mu_i \langle f, \phi_i \rangle^2_{\mathcal{H}}\,, \qquad \text{and} \qquad \lambda^2 \|h\|^2_{\mathcal{H}} = \sum\nolimits_i (\lambda + \mu_i)^2 \langle f, \phi_i \rangle^2_{\mathcal{H}}\,. \tag{5}$$

This reveals that P1 and P2 are connected through a *monotonic* spectral transformation. When λ is small, it simply squares the eigenvalues, which leads to little difference in learning as we observed in experiment. Moreover, there is a similar connection between c_x and the kernel representer of a new RKHS, which warps the original RKHS norm into $\|f\|^2_{\mathcal{H}} + R(f)$ [36]. See details in Appendix C.

ii) General R. When R is not quadratic, the linear relationship between c_x and k_x no longer exists. However, some relaxed connection between P1 and P2 is still available, and we will demonstrate it on dropout training. As discovered by [37, 38], dropout on input features in a single-layer network leads to an adaptive regularizer on a linear discriminant $x \mapsto \beta^\top x$ (derivation is in Appendix D):

$$R_{\tilde{p}}(\beta) = \sum\nolimits_i \mathbb{E}_{x \sim \tilde{p}}[p_x(1 - p_x)x_i^2] \cdot \beta_i^2, \quad \text{where} \quad p_x := \sigma(x^\top \beta) := (1 + \exp(-x^\top \beta))^{-1}. \tag{6}$$

Here $R_{\tilde{p}}$ penalizes β_i more mildly if x_i is generally small. This allows rare but discriminative features to receive higher weights, which is useful in text data. Now to connect P1 and P2, we simplify the

computation by using the proximal map of R_{δ_x} instead of $R_{\tilde{p}}$, where δ_x is the Dirac distribution at x:

$$c_x := \mathsf{P}_{R_{\delta_x}}(x) = \arg\min_c \left\{ \tfrac{1}{2} \sum\nolimits_i p_x(1-p_x) x_i^2 c_i^2 + \tfrac{\lambda}{2} \|c - x\|^2 \right\}, \text{ where } p_x = \sigma(x^\top c). \quad (7)$$

Since p_x depends only on $x^\top c$, we first fix $x^\top c$ to s, hence $p_x(1-p_x) = \alpha_s := \frac{2}{2+e^s+e^{-s}}$. Enforcing $x^\top c = s$ by a Lagrange multiplier μ, c_i's are decoupled, allowing them to be optimized analytically:

$$(c_x)_i = (\lambda + \mu) x_i (\lambda + \alpha_s x_i^2)^{-1}, \qquad \text{where} \quad \mu \text{ is such that } x^\top c_x = s. \quad (8)$$

Finally (7) can be optimized through a 1-D line search on s. Letting $h_i = \beta_i \frac{\lambda + \alpha_s x_i^2}{\lambda + \mu}$, we have $\beta^\top x = h^\top c_x$ (same predictions) and $\|h\|^2 = (\lambda+\mu)^{-2} \sum_i (\lambda + p_x(1-p_x) x_i^2)^2 \beta_i^2$, which resembles $R_{\tilde{p}}$ in (6) especially when λ is small. However, since $\frac{h_i}{\beta_i}$ depends on x, this reformulation meets with difficulty when extended to the whole dataset $\tilde{p}$. We emphasize that our aim here is to shed light on the connection between regularization and proximal mapping; we do *not* intend to establish their exact equivalence. Simulations in Appendix D show that P1 and P2 deliver very similar predictions.

Application to multiple layers. It is straightforward to apply proximal mapping to any hidden layer of interest and for multiple times. A similar warping trick was introduced in [36] to invariantize convolutional kernel descriptors [34, 39]. However it was restricted to linear invariances. Proximal mapping, instead, lifts this restriction by accommodating nonlinear invariances such as total variation.

4 Proximal Mapping for Robust Learning in Recurrent Neural Nets

Our first novel instance of ProxNet tries to invariantize LSTM to perturbations on inputs x_t. Virtual adversarial training has been proposed in this context as an unsupervised regularizer [40], where the underlying prior postulates that such robustness can benefit the prediction accuracy. The resilience under real attack, however, is *not* the main concern in [40]. We will demonstrate empirically that this prior can be more effectively implemented by ProxNet, leading to improved prediction performance.

The dynamics of hidden states c_t in an LSTM can be represented by $c_t = f(c_{t-1}, h_{t-1}, x_t)$, with outputs h_t updated by $h_t = g(c_{t-1}, h_{t-1}, x_t)$. We aim to encourage that the hidden state c_t stays invariant, when each x_t is perturbed by δ_t whose norm is bounded by δ. To this end, we introduce an intermediate step $s_t = s_t(c_{t-1}, h_{t-1}, x_t)$ that computes the original hidden state, and then apply proximal mapping so that the next state c_t remains close to s_t, while also moving towards the *null space* of the variation of s_t under the perturbations on x_t. Formally, using first-order approximation,

$$\begin{aligned} c_t :&= \arg\min_c \tfrac{\lambda}{2} \|c - s_t\|^2 + \tfrac{1}{2} \max\nolimits_{\|\delta_t\| \le \delta} \langle c, s_t(c_{t-1}, h_{t-1}, x_t) - s_t(c_{t-1}, h_{t-1}, x_t + \delta_t) \rangle^2 \\ &\approx \arg\min_c \lambda \|c - s_t\|^2 + \max\nolimits_{\|\delta_t\| \le \delta} \langle c, \tfrac{\partial}{\partial x_t} s_t(c_{t-1}, h_{t-1}, x_t) \delta_t \rangle^2 \\ &= \arg\min_c \lambda \|c - s_t\|^2 + \delta^2 \left\| c^\top G_t \right\|_*^2, \quad \text{where} \quad G_t := \tfrac{\partial}{\partial x_t} s_t(c_{t-1}, h_{t-1}, x_t) \end{aligned}$$

and $\|\cdot\|_*$ is the dual norm. The diagram is shown in Figure 2. Using the L_2 norm, a closed-form solution for c_t is $(I + \lambda^{-1} \delta^2 G_t G_t^\top)^{-1} s_t$, and BP can be reduced to second-order derivatives (§F). A key advantage of this framework is the generality and ease in inserting proximal layers into the framework — simply invoke the second-order derivatives of the underlying (gated) units as a black box, be it LSTM or GRU. We will refer to this model as ProxLSTM. To summarize, we achieved robustness not because of using the recurrent structure itself, but by properly invariantizing each step via embedding a proximal mapping, which is innately synergistic with the recurrent structure. So this technique can be generically deployed in neural networks.

5 ProxNet for Multiview Learning

While proximal mapping is applied on each individual data point in ProxLSTM, it can indeed be applied in mini-batches, and we next demonstrate its application in multiview learning with sequential structures. Here each instance exhibits a pair of views: $\{(x_i, y_i)\}_{i=1}^n$, and is associated with a label c_i. In the deep canonical correlation analysis model [DCCA, 10], the x-view is passed through a multi-layer neural network or kernel machine, leading to a hidden representation $f(x_i)$. Similarly the y-view is transformed into $g(y_i)$. CCA aims to maximize the correlation of these two views

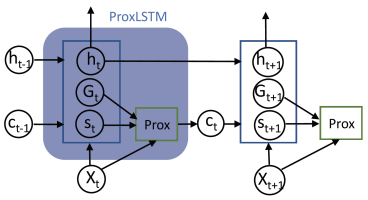


Figure 2: A proximal LSTM layer

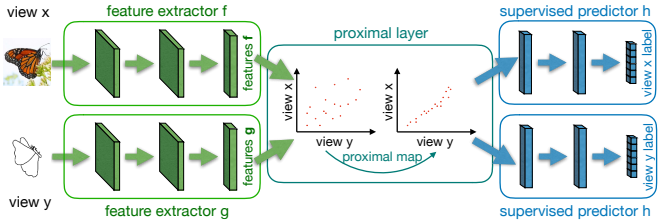


Figure 3: ProxNet for multiview learning with proximal CCA

after projecting into a common k-dimensional subspace, through $\{u_i\}_{i=1}^k$ and $\{v_i\}_{i=1}^k$ respectively. Denoting $X = (f(x_1), \ldots, f(x_n))H$ and $Y = (g(y_1), \ldots, g(y_n))H$ where $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ is the centering matrix, CCA finds $U = (u_1, \ldots, u_k)$ and $V = (v_1, \ldots, v_k)$ that maximize the correlation:

$$\min_{U,V} -\operatorname{tr}(U^\top XY^\top V), \tag{9}$$

$$\text{s.t. } U^\top XX^\top U = I,\ V^\top YY^\top V = I,\ u_i^\top XY^\top v_j = 0,\ \forall i \neq j. \tag{10}$$

Denote the optimal objective value as $L(X, Y)$. DCCA directly optimizes it with respect to the parameters in f and g, while DCCA autoencoder [DCCAE, 9] further reconstructs the input. They both use the result to initialize a finer tuning of f and g, in conjunction with subsequent layers h for a supervised target c_i. We aim to improve this two-stage process with an end-to-end approach based on proximal mapping, which can be written as $\min_{f,g,h} \sum_i \ell(h(p_i, q_i), c_i)$ where $\{(p_i, q_i)\}_{i=1}^n$ is from

$$\mathsf{P}_L(X, Y) = \arg\min_{P,Q} \tfrac{\lambda}{2n} \|P - X\|_F^2 + \tfrac{\lambda}{2n} \|Q - Y\|_F^2 + L(P, Q). \tag{11}$$

Here $\|\cdot\|_F$ stands for the Frobenius norm, $P = (p_1, \ldots, p_n)$, and $Q = (q_1, \ldots, q_n)$. Clearly, (11) has applied proximal mapping to a *mini-batch*, and we will show how to save computational cost, especially at test time. The entire framework is illustrated in Figure 3.

5.1 Backpropagation and computational cost

Although efficient closed-form solution is available for the CCA objective in (9), none exists for the proximal mapping in (11). However, it is natural to take advantage of this closed-form solution. In particular, assuming $f(x_i)$ and $g(y_i)$ have the same dimensionality, [10] showed that $L(X, Y) = -\sum_{i=1}^k \sigma_i(T)$, where σ_i is the i-th largest singular value, and

$$T(X, Y) = (XX^\top + \epsilon I)^{-1/2}(XY^\top)(YY^\top + \epsilon I)^{-1/2}.$$

Here $\epsilon > 0$ is a small stabilizing constant. Then (11) can be solved by gradient descent or L-BFGS. The gradient of $\sum_{i=1}^k \sigma_i(T(P, Q))$ is available from [10], which relies on SVD. Although SVD appears expensive, fortunately, the cost of computing T and SVD is low in practice because i) the dimensions of f and g are low in practice (10 in our experiment and DCCA), and ii) the mini-batch size does not need to be large. In our experiment, increasing mini-batch size beyond 100 did not significantly improve the performance. Extension to more than two views is relegated to Appendix E.

Backpropagation through the proximal mapping in (11) requires that given $\frac{\partial J}{\partial P}$ and $\frac{\partial J}{\partial Q}$ where J is the ultimate objective value, compute $\frac{\partial J}{\partial X}$ and $\frac{\partial J}{\partial Y}$. The most general solution has been provided by OptNet [14, 15], but the structure of our problem admits a simpler solution from [16].

$$\left(\tfrac{\partial J}{\partial X}, \tfrac{\partial J}{\partial Y}\right) \approx \tfrac{1}{\epsilon}\left(\mathsf{P}_L(X + \epsilon\tfrac{\partial J}{\partial P}, Y + \epsilon\tfrac{\partial J}{\partial Q}) - \mathsf{P}_L(X, Y)\right), \qquad 0 < \epsilon \ll 1. \tag{12}$$

[16] provided several heuristics for setting ϵ. We set $\epsilon = \delta(1 + ||(X, Y)||_\infty)||(\frac{\partial L}{\partial P}, \frac{\partial L}{\partial Q})||_\infty^{-1}$ in our experiments so as to be adaptive to the magnitude of the gradient, and δ is a small constant whose value was chosen by checking a few gradients at the beginning of training. To estimate the "true" gradient needed for the check, we used a small ϵ and solved the proximal mapping in (11) to high accuracy. With this heuristic, the approximate gradient did not cause instability in training. It is also noteworthy that the stochastic gradient computed from a mini-batch introduces noise in the first place.

To reduce the test time complexity for ProxNet, we draw a key insight that if the feature extractor preceding the proximal mapping is well trained so that the latent representation of the two views is highly correlated, then the proximal layer may improve performance only marginally.

Figure 4: ProxNet for multiview sequential data

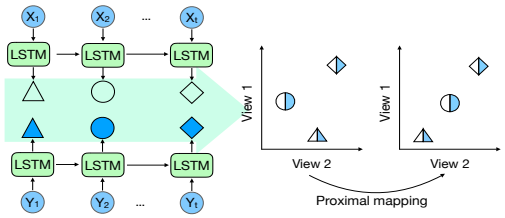


Table 1: Average test error (%) on Sketchy

#class	20	50	100	125
Vanilla	18.7 ± 1.1	24.8 ± 0.9	30.9 ± 0.5	31.8 ± 0.5
DCCA	16.9 ± 0.5	22.2 ± 0.4	28.7 ± 0.4	29.8 ± 0.4
DCCAE	16.6 ± 0.3	22.1 ± 0.3	29.2 ± 0.5	30.4 ± 0.6
RRM	15.2 ± 0.6	20.1 ± 0.4	26.8 ± 0.5	28.1 ± 0.4
RRM^{anl}	17.4 ± 0.8	22.5 ± 0.5	24.3 ± 0.9	26.1 ± 0.7
ProxNet	**13.7** ± 0.3	**17.9** ± 0.5	**20.2** ± 0.3	**22.0** ± 0.4

Therefore, we can take advantage of proximal mapping during training, while gradually fade it out at the fine tuning stage. Towards this end, the weight λ that controls the trade off between correlation and displacement can be increased as training proceeds. More specifically, we set in experiment $\lambda_t = (1 + kt)\alpha_0$ at epoch t, where α_0 and k are hyperparameters. As a result, test time predictions can be made very efficiently by dispensing with proximal mapping or mini-batch.

6 Experimental Results

We evaluated the empirical performance of ProxNet for multiview learning on supervised learning (two tasks) and unsupervised learning (crosslingual word embedding). ProxLSTM was evaluated on sequence classification. We used the Ray Tune library to select the hyper-parameters for all baseline methods [41]. Details on data preprocessing, experiment setting, optimization, and additional results are given in Appendix G. Here we highlight the major results and experiment setup.

All code and data are available at `https://github.com/learndeep2019/ProxNet`.

Baselines. For the three multiview tasks, we will demonstrate the effectiveness of ProxNet by comparing with state-of-the-art methods including **DCCA** and **DCCAE**. Neither DCCA nor DCCAE is end-to-end training, and a classifier was trained on their hidden code. As a basic competitor, we also considered a **Vanilla** method, which trained a network for each view independently.

Our key competitor is RRM, which motivated ProxNet in the introduction section. Specifically, it moves the regularizer $L(X, Y)$ defined in (9) from inside the proximal mapping to the overall objective as in (1), promoting the correlation between the two views' hidden representation through the network weights. At test time, ProxNet, RRM, and Vanilla all made predictions by averaging the logits from both views. This consistently outperformed concatenating the logits of the two views.

6.1 Multiview supervised learning 1: image recognition with sketch and photo

Dataset. We first evaluated ProxNet on a large-scale sketch-photo paired database "Sketchy" [42]. It consists of 12,500 photos and 75,471 hand-drawn sketches of objects from 125 classes. Each instance is a pair of sketch and photo representing the same natural image, both in color and sized 256×256. To demonstrate the robustness of our method, we varied the number of classes over $\{20, 50, 100, 125\}$ by sampling a subset from the original dataset. For each class, there are 100 sketch-photo pairs. We randomly sampled 80 pairs from each class to form the training set, and then used the remaining 20 pairs for testing.

Implementation details. Unless otherwise specified, our implementations were based on PyTorch and all training was conducted on a NVIDIA GeForce 2080 Ti GPU. All methods were trained using ResNet-18 as the feature extractor. In ProxNet, the proximal layer has input and output dimension $d = 20$, followed by three fully-connected layers of 512 hidden units with sigmoid activations. The final output layer has multiple softmax units, each corresponding to an output class. ProxNet was trained by Adam with a weight decay of 0.0001 and a learning rate of 0.001, with the latter divided by 10 after 200 epochs. The mini-batch size was 100, which, in conjunction with the low dimensionality of proximal layer ($d = 20$), allows the SVD to be solved instantaneously. At training time, we employed an adaptive trade-off parameter λ, which is defined in (11). We set the hyper-parameters $k = 0.5$ and $\alpha_0 = 0.1$. All experiments were run five times to produce mean and standard deviation.

Table 2: Mean and standard deviation of PERs on the XRMB dataset with different noise levels

	noise level = 0%		noise level = 20%		noise level = 50%		noise level = 80%	
	acoustic	logit-avg	acoustic	logit-avg	acoustic	logit-avg	acoustic	logit-avg
Vanilla	17.9 $\pm$ 1.0	17.1 $\pm$ 0.6	19.3 $\pm$ 0.8	19.1 $\pm$ 1.2	27.7 $\pm$ 1.1	21.4 $\pm$ 0.8	45.1 $\pm$ 1.0	24.4 $\pm$ 1.0
DCCA	17.3 $\pm$ 0.3	16.3 $\pm$ 0.5	18.8 $\pm$ 0.3	16.3 $\pm$ 0.6	26.0 $\pm$ 0.3	23.6 $\pm$ 0.5	45.1 $\pm$ 0.9	34.9 $\pm$ 0.9
DCCAE	15.5 $\pm$ 0.2	15.3 $\pm$ 0.4	16.7 $\pm$ 0.3	15.9 $\pm$ 0.4	23.6 $\pm$ 0.3	21.8 $\pm$ 0.7	43.9 $\pm$ 0.7	34.8 $\pm$ 0.7
RRM	16.1 $\pm$ 0.5	15.0 $\pm$ 0.3	16.6 $\pm$ 0.7	16.9 $\pm$ 0.5	**22.3** $\pm$ 0.8	21.6 $\pm$ 0.6	40.7 $\pm$ 0.7	23.9 $\pm$ 0.3
ProxNet	**12.9** $\pm$ 0.4	**10.5** $\pm$ 0.4	**15.3** $\pm$ 0.4	**11.2** $\pm$ 0.3	**21.6** $\pm$ 0.5	**16.6** $\pm$ 0.3	**39.3** $\pm$ 0.3	**20.1** $\pm$ 0.5

Results. As shown in Table 1, ProxNet delivers significantly lower test error than all other baselines. Interestingly, the improvement becomes more significant with the increasing number of classes. Vanilla performs the worst, and RRM outperforms DCCA and DCCAE thanks to end-to-end training.

Since we annealed the weight λ in the proximal mapping (11) (Section 5.1), it is natural to investigate whether an annealed regularization weight can improve the performance of RRM. Therefore, we conducted additional experiments for it, and the resulting average test error is also presented in Table 1 as RRM^{anl}. Clearly, the influence on the vanilla RRM is mixed, but it remains inferior to ProxNet.

6.2 Multiview supervised learning 2: audio-visual speech recognition

Our second task aims to learn features and classifiers for speaker-independent phonetic recognition.

Dataset. We used the Wisconsin X-ray Micro-Beam Database (XRMB) corpus which consists of simultaneously recorded *speech* and *articulatory* measurements from 47 American English speakers and 2357 utterances [43]. The first view is acoustic features comprising 39D mel frequency cepstral coefficients (MFCCs) and their first and second derivatives, and the second view is articulatory features made up of 16D horizontal/vertical displacement of 8 pellets attached to several parts of the vocal tract. Also available is the phonetic labels for classification. To simulate the real-life scenarios and to improve the model's robustness to noise, we corrupted the acoustic features of a given speaker by mixing with $\{0.2, 0.5, 0.8\}$ level of another randomly picked speaker's acoustic features. The whole dataset was partitioned into 35 speakers for training and 12 speakers for testing.

Implementation details. To incorporate context information, [9] concatenated the inputs over a window sized W centered at each frame, giving $39 \times W$ and $16 \times W$ feature dimensions for each of the two views respectively. Although this delicately constructed input freed the encoder/feature extractor from considering the time dependency within frames, we prefer a more refined modeling of the sequential structure, and therefore adopted, for all methods under comparison, a 2-layer LSTM with hidden layers of 256 units, followed by a fully-connected layer which projects the outputs of LSTM to a K-dimensional subspace, serving as the feature extractor.

The supervised predictor is a fully-connected network with an output layer of 41 softmax units. We used the Connectionist Temporal Classification loss [CTC, 44], which adopts greedy search as the phone recognizer. The dimension of subspace was tuned in $\{10, 20, 30, 50\}$, and the sequence length was tuned in $\{250, 500, 1000\}$ for all algorithms. The mini-batch size was set to 32. Although the proximal mapping here solves a larger problem than that in §6.1, we observed that a higher value of λ was sufficient to enforce a high correlation on this dataset, hence keeping the optimization efficient. In practice, we set $k = 1$ and $\alpha_0 = 0.5$.

In order to compare the effectiveness of different algorithms in information transfer without being confounded by logit averaging (logit-avg) which can achieve a similar effect, we studied another mode called "acoustic". Here all algorithms predict on test data by only using the output layer of the acoustic view, and at training time a loss is applied to each view based on the ground truth label.

Results. Table 2 presents the Phone Error Rates (PERs) of all methods. Clearly, ProxNet achieves the lowest PER among all algorithms at all levels of noise. The margin over the runner-up (RRM) is the largest when there is no noise. As expected, "logit-avg" almost always outperforms "acoustic", because the articulatory features are clean, supplying reliable predictions. Focusing on the "acous-

Table 3: Spearman's correlation for word similarity. Following [45], for each algorithm, the model with the highest Spearman's correlation on the 649 tuning bigram pairs was selected.

	WS-353		WS-SIM		WS-REL		SimLex999	
	EN	DE	EN	DE	EN	DE	EN	DE
Baseline	73.4	52.7	77.8	63.3	67.7	44.2	37.2	29.1
LinCCA	73.8	68.5	76.1	73.0	67.0	62.9	37.8	43.3
DCCA	73.9	69.1	**78.7**	74.1	66.6	64.7	38.78	43.29
DCCAE / RRM	72.4	**69.7**	75.7	74.7	65.9	64.2	36.7	41.8
ProxNet	**75.4**	69.2	78.3	**75.4**	**71.0**	**66.8**	**40.0**	**44.2**
CL-DEPEMB	-	-	-	-	-	-	35.6	30.6

tic" columns, Vanilla cannot leverage articulatory features, while other methods can achieve it by promoting correlations in the hidden space. ProxNet appears most effective in this respect.

6.3 Multiview unsupervised learning: crosslingual word embedding

We next seek to learn word representations that reflect word similarity, and the multiview approach trains on (English, German) word pairs, hoping that information is transferred in the latent subspace.

Dataset. We obtained 36K pairs of English-German word as training examples from the parallel news commentary corpora [WMT 2012-2018, 46], using the word alignment method from [47] and [48]. Based on the corpora we also built a bilingual dictionary, where each English word is matched with the (unique) German word that has been most frequently aligned to it. The raw word embedding (x_i and y_i) used the pretrained monolingual 300-dimensional word vectors from fastText [49, 50].

The evaluation was conducted on two commonly used datasets [51, 52]: **a)** multilingual WS353 contains 353 pairs of English words, and their translations to German, Italian and Russian, that have been assigned similarity ratings by humans. It was further split into multilingual WS-SIM and multilingual WS-REL which measure similarity and relatedness between word pairs, respectively; **b)** multilingual SimLex999 consists of 999 English word pairs and their translations.

Algorithms. All methods used multilayer perceptrons with ReLU activation. ProxNet used the input reconstruction error as the ultimate objective. As a result, *DCCAE is exactly the RRM variant*. A validation set was employed to select the hidden dimension h for f and g from $\{0.1, 0.3, 0.5, 0.7, 0.9\} \times 300$, the regularization parameter λ, and the depth and layer width from 1 to 4 and $\{256, 512, 1024, 2048\}$, respectively. We searched the mini-batch size in $\{100, 200, 300, 400\}$. We also compared with linear CCA [53]. At test time, the (English, German) word pairs from the test set were fed to the four multiview based models, extracting the English and German word representations. Then the cosine similarity can be computed between all pairs of monolingual words in the test set (English and German), and we reported in Table 3 the Spearman's correlation between the model's ranking and human's ranking.

Results. Clearly, ProxNet always achieves the highest or close to highest Spearman's correlation on all test sets and for both English (EN) and German (DE). We also included a baseline which only uses the monolingual word vectors. CL-DEPEMB is from [54], and the paper only provided the results for SimLex999 with no code made available. It can be observed from Table 3 that multiview based methods achieved more significant improvement over the baseline on German data than on English data. This is not surprising, because the presence of multiple views offers an opportunity to transfer useful information from other views/languages. Since the performance on English is generally better than that of German, more improvement is expected on German.

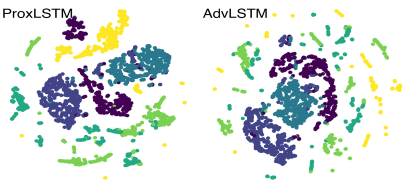
6.4 Robust training for recurrent networks

We now present the experimental results of robust training for LSTMs as described in Section 4.

Table 4: Test accuracy for sequence classification. "len" stands for the median length of the sequences.

	#train	len	LSTM	AdvLSTM	ProxLSTM
JV	225	15	94.02 ±0.72	94.96 ±0.44	**95.52** ±0.63
HAR	6.1k	128	89.75 ±0.59	**92.01** ±0.18	**92.08** ±0.23
AD	5.5k	39	96.32 ±0.55	97.45 ±0.38	**97.99** ±0.29
IMDB	25k	239	92.65 ±0.04	93.65 ±0.03	**94.16** ±0.11

Figure 5: t-SNE embedding of HAR dataset (best viewed in color)



Datasets. We tested on four sequence datasets: Japanese vowels [JV, 55] which contains time series data for speaker recognition based on uttered vowels; Human Activity Recognition [HAR, 56] which classifies activity; Arabic Digits [AD, 57] which recognizes digits from speeches; and IMDB [58], a large movie review dataset for sentiment classification. Table 4 presents the training set size and *median* sequence length.

Algorithms. We compared ProxLSTM with two baselines: vanilla LSTM and the adversarial training of LSTM [40], which we will refer to as AdvLSTM. For JV, HAR, AD datasets, the base models are preceded by a CNN layer, and succeeded by a fully connected layer. The CNN layer consists of kernels sized 3, 8, 3 and contains 32, 64, 64 filters for the JV, HAR, AD datasets, respectively. LSTM used 64, 128, 64 hidden units for these three datasets, respectively. All these parameters were tuned to optimize the performance of vanilla LSTM, and then shared with ProxLSTM and AdvLSTM for a fair comparison. We first trained the vanilla LSTM to convergence, and used the resulting model to initialize AdvLSTM and ProxLSTM. For IMDB, we first trained AdvLSTM by following the settings in [40], and then used the result to initialize the weights of ProxLSTM. All settings were evaluated 10 times to report mean and standard deviation.

Results. From Table 4, it is clear that adversarial training improves test accuracy, and ProxLSTM promotes the performance even more than AdvLSTM. Since the accuracy gap is lowest on the HAR dataset, we also plotted the t-SNE embedding of the features from the *last time step* for HAR. As Figure 5 shows, the representation learned by ProxLSTM is better clustered than that of AdvLSTM, especially the yellow class. This further indicates that ProxLSTM learns better latent representations than AdvLSTM by applying proximal mapping. Plots for other datasets are in §G.4.

7 Conclusion

In this paper, we proposed using proximal mapping as a new primitive in deep networks to explicitly encode the prior for end-to-end training. Connection to existing constructs in deep learning are shown. The new model is extended to multiview learning and robust RNNs, and its effectiveness is demonstrated in experiments. It is noteworthy that ProxNet is a means of enforcing deep regularization, while itself does not introduce any new regularizer per se; the regularizer R is to be designed by the practitioners for the specific application, e.g., CCA for multi-view learning. Implementing ProxNet is straightforward as shown in (2) for any generic R, though some parameters need to be chosen.

The purpose of the paper is to show that for regularizers defined in terms of hidden layer outputs, it is more effective to regularize **in-place** through a proximal mapping at that layer, compared with adding the regularizer to the overall objective and relying on backpropagation for optimization. By "more effective", we have compared by using the test performance instead of the training objective value, because unlike comparing two different nonconvex solvers, ProxNet results in a different objective than regularized risk minimization.

A rigorous analysis beyond the intuition of in-place regularization will be interesting in the deep context, and we plan to investigate it in the future. We will also apply ProxNet to reinforcement learning with knowledge transfer. Furthermore, it will be very interesting to study the use of multiple proximal mappings at different layers for diverse purposes, e.g., to enforce equivariance in each layer of feature extractor by using the violation as the regularizer R, disentanglement at a certain layer, and fairness in prediction.

Acknowledgements

We thank the reviewers for their constructive comments. This work is supported by Google Cloud and NSF grant RI:1910146. The experiments used the Extreme Science and Engineering Discovery Environment (XSEDE) at the PSC GPU-AI. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Broader Impact

Information can be presented in multiple sensory modalities like vision, sound, and touch. However, many machine/deep learning algorithms are still trained on single-modality data instead of taking full advantage of multiple modalities. Recent works have shown that these applications learning from single-modality data might imperil the model by producing biased and/or even unfair results. For instance, a model trained on image data, which has little or no effect on the acoustic and tactile properties of the imaged scene, might has a different or opposite understand of its semantics which results in a wrong prediction [59]. Our work aims to mitigate this problem by learning representations that capture information shared between two views. Thanks to its flexibility and efficiency of ProxNet, this technique can also be extended to many other applications that may affect our daily life.

Temporal multiview learning for brain network analysis and mortality forecasting. In neurological disorder analysis, fMRI and diffusion tensor imaging provide different views of the same brain, and each of them represents a graph of brain regions, evolving over the duration of scanning. This evolution can be modeled as dynamic graphs. Naturally, we can apply the multiview ProxNet to graph convolutional networks to discover salient representations that provide insightful interpretations for medical practitioners. Similar techniques also can be used to understand mortality rates from multiple populations (views) with a time-series structure.

Adversarial recurrent networks for sentiment analysis on reviews. In sentiment analysis of reviews, a writer can outwit the auto-rater by changing the style, such as inserting common words and avoiding specific keywords consistently over time. Interestingly, invariance to such perturbations in a sequential model of text can be effectively modeled by inserting a proximal layer at each step to build adversarial ProxNet based on LSTMs.

References

[1] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *In International Conference on Learning Representations (ICLR)*. 2017.

[2] D. Arpitz, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A closer look at memorization in deep networks. *In International Conference on Machine Learning (ICML)*. 2017.

[3] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. *In* G. Montavon, G. B. Orr, and K.-R. Müller, eds., *Neural Networks: Tricks of the Trade: Second Edition*, pp. 235–269. Springer, 2012.

[4] D. K. Pal, A. A. Kannan, G. Arakalgud, and M. Savvides. Max-margin invariant features from transformed unlabeled data. *In Advances in Neural Information Processing Systems (NIPS)*. 2017.

[5] A. Makhzani and B. Frey. k-Sparse autoencoders. *In International Conference on Learning Representations (ICLR)*. 2014.

[6] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Adversarially robust training through structured gradient regularization, 2018. ArXiv:1805.08736.

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[8] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. *In International Conference on Machine Learning (ICML)*. 2011.

[9] W. Wang, R. Arora, K. Livescu, and J. A. Bilmes. On deep multi-view representation learning. *In International Conference on Machine Learning (ICML)*. 2015.

[10] G. Andrew, R. Arora, J. A. Bilmes, and K. Livescu. Deep canonical correlation analysis. *In International Conference on Machine Learning (ICML)*. 2013.

[11] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. *In International Conference on Machine Learning (ICML)*. 2019.

[12] M. Carreira-Perpinnan and W. Wang. Distributed optimization of deeply nested systems. *In International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2014.

[13] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[14] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. *In International Conference on Machine Learning (ICML)*. 2017.

[15] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *In Advances in Neural Information Processing Systems (NIPS)*. 2019.

[16] J. Domke. Implicit differentiation by perturbation. *In Advances in Neural Information Processing Systems (NIPS)*. 2010.

[17] J. Domke. Generic methods for optimization-based modeling. *In International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2012.

[18] D. Belanger, B. Yang, and A. McCallum. End-to-end learning for structured prediction energy networks. *In International Conference on Machine Learning (ICML)*. 2017.

[19] B. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[20] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *In International Conference on Learning Representations (ICLR)*. 2017.

[21] L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. *In International Conference on Learning Representations (ICLR)*. 2019.

[22] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[23] B. Amos, I. Rodriguez, J. Sacks, B. Boots, and J. Kolter. Differentiable mpc for end-to-end planning and control. *In Advances in Neural Information Processing Systems (NIPS)*. 2018.

[24] I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio. Multi-prediction deep boltzmann machines. *In Advances in Neural Information Processing Systems (NIPS)*. 2013.

[25] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv:1607.05447*, 2016.

[26] V. Niculae and M. Blondel. A regularized framework for sparse and structured neural attention. *In Advances in Neural Information Processing Systems (NIPS)*. 2017.

[27] V. Niculae, A. F. Martins, M. Blondel, and C. Cardie. Sparsemap: Differentiable sparse structured inference. *In International Conference on Machine Learning (ICML)*. 2018.

[28] A. Martins and R. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *In International Conference on Machine Learning (ICML)*. 2016.

[29] P. L. Combettes and J. C. Pesquet. Deep neural network structures solving variational inequalities. *arXiv:1808.07526*, 2018.

[30] W. Hare and C. Sagastizábal. Computing proximal points of nonconvex functions. *Mathematical Programming*, 116(1):221–258, 2009.

[31] F. Bernard and L. Thibault. Prox-regularity of functions and sets in banach spaces. *Set-Valued Analysis*, 12(1):25–47, Mar 2004.

[32] R. A. Poliquin and R. T. Rockafellar. Prox-regular functions in variational analysis. *Transactions of the American Mathematical Society*, 348(5):1805–1838, 1996.

[33] P. Laforgue, S. Clémençon, and F. d'Alché-Buc. Autoencoding any data through kernel autoencoders. *In International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019.

[34] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. *In Advances in Neural Information Processing Systems (NIPS)*. 2014.

[35] A. Bietti and J. Mairal. Group Invariance, Stability to Deformations, and Complexity of Deep Convolutional Representations. *In Advances in Neural Information Processing Systems (NIPS)*. 2017.

[36] Y. Ma, V. Ganapathiraman, and X. Zhang. Learning invariant representations with kernel warping. *In International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2019.

[37] S. Wager, S. I. Wang, and P. Liang. Dropout training as adaptive regularization. *In Advances in Neural Information Processing Systems (NIPS)*. 2013.

[38] S. I. Wang and C. D. Manning. Fast dropout training. *In International Conference on Machine Learning (ICML)*. 2013.

[39] J. Mairal. End-to-end kernel learning with supervised convolutional kernel networks. *In Advances in Neural Information Processing Systems (NIPS)*. 2016.

[40] T. Miyato, A. M. Dai, and I. Goodfellow. Adversarial training methods for semi-supervised text classification. *In International Conference on Learning Representations (ICLR)*. 2017.

[41] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[42] P. Sangkloy, N. Burnell, C. Ham, and J. Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016.

[43] J. R. Westbury. X-ray microbeam speech production database user's handbook version 1.0. *Tech. rep.*, Waisman Center on Mental Retardation and Human Development, University of Wisconsin, 1994.

[44] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *In ICML '06*. 2006.

[45] W. Wang, R. Arora, K. Livescu, and J. Bilmes. On deep multi-view representation learning. *In International Conference on Machine Learning (ICML)*. 2015.

[46] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, M. Huck, A. J. Yepes, P. Koehn, C. Monz, M. Negri, A. Neveol, M. Neves, M. Post, L. Specia, M. Turchi, K. Verspoor, and M. Fishel. News commentary corpus. 2018. `http://www.statmt.org/wmt18`.

[47] C. Dyer, V. Chahuneau, and N. A. Smith. A simple, fast, and effective reparameterization of ibm model 2. *In HLT-NAACL*. 2013.

[48] FastAlign. Fast align toolbox. `https://github.com/clab/fast_align`.

[49] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov. Learning word vectors for 157 languages. *In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[50] FastText. Pretrained fasttext word vectors. https://fasttext.cc/docs/en/crawl-vectors.html.

[51] I. Leviant and R. Reichart. Separated by an un-common language: Towards judgment language informed vector space modeling. *arXiv:1508.00106*, 2015.

[52] I. Leviant and R. Reichart. Multilingual simlex999 and wordsim353 datasets, 2019. http://leviants.com/ira.leviant/MultilingualVSMdata.html.

[53] M. Faruqui and C. Dyer. Improving vector space word representations using multilingual correlation. *In EACL*. 2014.

[54] I. Vulić. Cross-lingual syntactically informed distributed word representations. *In European Chapter of the Association for Computational Linguistics*. 2017.

[55] M. Shimbo, M. Kudo, and J. Toyama. Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20:1103, 1999.

[56] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. *In European Symposium on Artificial Neural Networks*. 2013.

[57] N. Hammami and M. Bedda. Improved tree model for arabic speech recognition. *In International Conference on Computer Science and Information Technology*. 2010.

[58] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. *In Association for Computational Linguistics (ACL)*, pp. 142–150. Association for Computational Linguistics, 2011.

[59] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *In International Conference on Learning Representations (ICLR)*. 2015.

[60] P. Brakel, D. Stroobandt, and B. Schrauwen. Training energy-based models for time-series imputation. *Journal of Machine Learning Research*, 14:2771–2797, 2013.

[61] V. Stoyanov, A. Ropson, and J. Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. *In International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

[62] J. A. Bagnell and D. M. Bradley. Differentiable sparse coding. *In Advances in Neural Information Processing Systems (NIPS)*. 2008.

[63] J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):791–804, 2012.

[64] T. Meinhardt, M. Moeller, C. Hazirbas, and D. Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. *In International Conference on Computer Vision (ICCV)*. 2017.

[65] A. Rajeswaran, C. Finn, S. Kakade, and S. Levine. Meta-learning with implicit gradients. *In Advances in Neural Information Processing Systems (NIPS)*. 2019.

[66] S. Wang, S. Fidler, and R. Urtasun. Proximal deep structured models. *In Advances in Neural Information Processing Systems (NIPS)*. 2016.

[67] J. T. Zhou, K. Di, J. Du, X. Peng, H. Yang, S. J. Pan, I. W.-H. Tsang, Y. Liu, Z. Qin, and R. S. M. Goh. Sc2net: Sparse lstms for sparse coding. *In National Conference of Artificial Intelligence (AAAI)*. 2018.

[68] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. *In International Conference on Machine Learning (ICML)*. 2015.

[69] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *In Advances in Neural Information Processing Systems (NIPS)*. 2016.

[70] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. *In Advances in Neural Information Processing Systems (NIPS)*. 2017.

[71] C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *In International Conference on Learning Representations (ICLR)*. 2018.

[72] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *In International Conference on Machine Learning (ICML)*. 2017.

[73] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. *In International Conference on Artificial Neural Networks*. 2001.

[74] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. *In Advances in Neural Information Processing Systems (NIPS)*. 2016.

[75] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *In International Conference on Learning Representations (ICLR)*. 2017.

[76] A. Bietti and J. Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *Journal of Machine Learning Research*, 20(25):1–49, 2019.

[77] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. *In Advances in Neural Information Processing Systems (NIPS)*. 2000.

[78] P. Horst. Generalized canonical correlations and their applications to experimental data. *Journalof Clinical Psychology*, 17(4), 1961.

[79] P. Rastogi, B. V. Durme, and R. Arora. Multiview LSA: Representation learning via generalized CCA. *In Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2015.

[80] A. Benton, H. Khayrallah, B. Gujral, D. A. Reisinger, S. Zhang, and R. Arora. Deep generalized canonical correlation analysis. *In Workshop on Representation Learning for NLP*. 2019.

[81] A. Lu, W. Wang, M. Bansal, K. Gimpel, and K. Livescu. Deep multilingual correlation for improved word embeddings. *In Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2015.

Supplementary Material

A Relationship with OptNet and Implicit Differentiation Based Learning

Given a prediction model such as linear model, energy-based model, kernel function, deep neural network, etc, a loss function is needed to measure the quality of its prediction against the given ground truth. Although surrogate losses had been popular in making the loss convex, recently it is often observed that directly comparing the prediction of the model, typically computed through an argmin optimization (or argmax), against the ground truth under the true loss of interest can be much more effective. The error signal is originated from the *last step* through the argmin, and then backpropagated through the model itself for training. For example, Amos et al used it to train input convex neural networks at ICML 2017, [18] used it to train a structured prediction energy network, and [60] used it to train an energy-based model for time-series imputation. Other works include [24, 61], etc. A number of implicit and auto-differentiation algorithms have been proposed for it, e.g., [14, 16, 17, 19, 25].

Other uses of such differentiable optimization have been found in learning attention models [26], meta-learning to differentiate through the base learning algorithm [21, 22], or to train the generator in a generative adversarial model by optimizing out the discriminator [20], or for end-to-end planning and control [23]. In all these cases, differentiable optimization is used as an algorithm to train a *given* component within a multi-component learning paradigm. But each component itself has its own pre-fixed model and parameterization.

To the best of our knowledge, OptNet [14] proposed for the first time using optimization as a *layer* of the deep neural network, hence extending the model itself. However, it focused on efficient algorithms for differentiation[1] and the general framework of optimization layer was demonstrated by using standard operations such as total variation denoising, which bears resemblance to task-driven dictionary learning [62, 63]. It remains unclear how to leverage the general framework of OptNet to flexibly model a broad range of structures, while reusing the existing primitives in deep learning (like our extension of LSTM in Section 4).

This is achieved by ProxNet. Although ProxNet also inserts a new layer, it provides *concrete and novel* ways to model structured priors in data through proximal mapping. [64] used proximal operators for regularizing inverse imaging problems. Most aforementioned works use differentiable optimization as a learning algorithm for a *given* model, while ProxNet uses it as a first-class modeling construct within a deep network. Designing the potential function f in (2) can be highly nontrivial, as we have demonstrated in the examples of dropout, kernel warping, multiview learning, and LSTM.

[65] used proximal mapping for the inner-level optimization of meta-learning, which constitutes a bi-level optimization. Their focus is to streamline the optimization using implicit gradient, while our goal, in contrast, is to use proximal mapping to learn structured data representations.

We note that despite the similarity in titles, [66] differs from our work as it applies proximal mapping in a solver to perform inference in a graphical model, whose cliques are neural networks. The optimization process *happens to* be analogous to a recurrent net, interlaced with proximal maps, and similar analogy has been drawn between the ISTA optimization algorithm and LSTM [67]. We instead use proximal map as a first-class construct/layer in a deep network.

B Connecting ProxNet with Meta-learning

In view that ProxNet is a bi-level optimization and the z in (2) may consist of the embeddings of input objects in *mini-batches*, we can interpret ProxNet from a meta-learning perspective. In particular, each mini-batch corresponds to a "task" (or dataset, episode, etc) in the standard meta-learning terminology, and the regularization term corresponds to the task-specific base learner inside each episode of the meta learner. Naturally, the preceding layers serve as the meta-parameters subject to meta-learning. For example, [68–70] used simple metric-based nearest neighbor, [71, 72] optimized

[1] Although the original paper only detailed on quadratic optimization mainly for the efficient GPU implementation, it is conceptually applicable to general nonlinear optimization. Such extensions have been achieved in [15].

standard learning algorithms iteratively, and [21, 22] leveraged closed-form solutions for base learners. Explicit learning of learner's update rule was investigated in [73–75]. In this sense, ProxNet extends meta-learning to *unsupervised* base learners.

We emphasize that ProxNet only leverages the idea and technique in meta-learning. It is beyond our paper to address existing challenges in meta-learning itself.

Detailed description The conventional meta-learning has a meta-parameter p, and each base-learner (for each task) has its own base-parameters w. Then by Equation (1) of the paper

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, Sergey Levine. *Meta-Learning with Implicit Gradients*. Neural Information Processing Systems (NeurIPS), 2019,

the bi-level optimization in meta-learning can be set up as ("perf" for "performance"):

$$\min_p \sum_i \text{Test-perf}\left(\arg\min_w \text{Training-perf}(w, p, \mathcal{D}_i^{train}), p, \mathcal{D}_i^{test}\right). \tag{13}$$

Here $\mathcal{D}_i^{train}$ and $\mathcal{D}_i^{test}$ are the training and test data for task i, respectively. Now we can establish the one-to-one correspondence between (13) and ProxNet in the context of multiview learning. Please refer to Section 5 for notations, especially Equations (9) and (11).

- p: the union of i) the feature extractors f and g for the two views, and ii) the downstream supervised layers. Only the former (f and g) is used in the inner training ($\arg\min_w$), which transforms the raw data into the input of the proximal layer.
- w: the U and V projection directions used by CCA;
- $\mathcal{D}_i^{train}$: the i-th mini-batch $\{(x_i, y_i)\}_{i=1}^n$;
- Training-perf$(w, p, \mathcal{D}_i^{train}) = \min_{P,Q} \frac{\lambda}{2n} \|P - X\|_F^2 + \frac{\lambda}{2n} \|Q - Y\|_F^2 - \text{tr}(U^\top P Q^\top V)$, where $X = (f(x_1), \ldots, f(x_n))$ and $Y = (g(y_1), \ldots, g(y_n))$. That is, for any given projection directions U and V (i.e., w), what is the minimal denoising objective, which combines the displacement (Frobenius norm) and the CCA objective (correlation between the projections);
- $\mathcal{D}_i^{test}$: the i-th mini-batch (same as $\mathcal{D}_i^{train}$);
- Test-perf: pass $\mathcal{D}_i^{test}$ through f and g, followed by denoising based on the trained $w = (U, V)$: $\arg\min_{P,Q} \frac{\lambda}{2n} \|P - X\|_F^2 + \frac{\lambda}{2n} \|Q - Y\|_F^2 - \text{tr}(U^\top P Q^\top V)$, and finally apply the supervised layers to measure the test performance.

So ProxNet effectively corresponds to a base-learner of multiview denoising. It extends the common meta-learning practice in two ways:

- the base-learner is unsupervised;
- the training and test performance employ different tasks (denoising versus error).

The latter is quite a valid learning paradigm: the training phase extracts useful representations as parameterized by U and V, and then the product (U and V) is evaluated on the test data by computing their projections, followed by a supervised loss. Since mini-batch sizes are very small (also intended to keep the optimization efficient), it can be considered as a few-shot learning. Surely the algorithm does not have to be restricted to mini-batches that are drawn iid; different mini-batches can employ bona-fide different learning tasks.

C Connecting Proximal Mapping to Kernel Warping

The graph Laplacian on a function f is $\sum_{ij} w_{ij}(f(x_i) - f(x_j))^2$, where $f(x_i) - f(x_j)$ is bounded and linear in f. Parameterizing an image as $I(\alpha)$ where α is the degree of rotation/translation/etc, transformation invariance favors a small magnitude of $\frac{\partial}{\partial \alpha}|_{\alpha=0} f(I(\alpha))$, again a bounded linear functional. By Riesz representation theorem, a bounded linear functional can be written as $\langle z_i, f \rangle_{\mathcal{H}}$

for some $z_i \in \mathcal{H}$. We will refer to z_i as an invariance representer, and suppose we have m such invariances.

In order to respect the desired invariances, [36] proposed a warped RKHS $\mathcal{H}^\circ$ consisting of the same functions in the original $\mathcal{H}$, but redefining the norm and the corresponding kernel by

$$\|f\|_{\mathcal{H}^\circ}^2 := \|f\|_{\mathcal{H}}^2 + \sum\nolimits_{i=1}^m \langle z_i, f\rangle_{\mathcal{H}}^2 \tag{14}$$

This leads to a new RKHS consisting of the same set of functions as $\mathcal{H}$, but its inner product warped into

$$\langle f, g\rangle_{\mathcal{H}^\circ} := \langle f, g\rangle_{\mathcal{H}} + \sum\nolimits_{i=1}^m \langle f, z_i\rangle_{\mathcal{H}} \langle g, z_i\rangle_{\mathcal{H}}\,, \tag{15}$$

and its kernel is warped into

$$k^\circ(x_1, x_2) = k(x_1, x_2) - z(x_1)^\top K_Z z(x_2), \tag{16}$$

where $z(x) = (z_1(x), \ldots, z_m(x))^\top$. Then replacing $k(x, \cdot)$ by $k^\circ(x, \cdot)$ results in a new invariant representation. Such a warping can be applied to all layers in, e.g., deep convolutional kernel networks [CKNs, 76], instilling invariance with respect to preceding layer's output.

The major limitation of this method, however, is that the invariances have to be modeled by the square of a linear form — $\langle z_i, f\rangle_{\mathcal{H}}^2$ — in order to make $\|f\|_{\mathcal{H}}^2 + \sum_{i=1}^m \langle z_i, f\rangle_{\mathcal{H}}^2$ a norm square, precluding many interesting invariances such as total variation $f \mapsto \int |f'(x)|\,\mathrm{d}x$.

Interestingly, this can be achieved by simply reformulating kernel warping as proximal mapping. To this end, recall that a Euclidean embedding maps $f \in \mathcal{H}$ to a real vector $\tilde{f}$, such that $\langle \tilde{f}, \tilde{h}\rangle \approx \langle f, h\rangle_{\mathcal{H}}$ for all $f, h \in \mathcal{H}$. A commonly used formula for embedding is the Nyström approximation [77]. Using p samples $W := \{\omega_i\}_{i=1}^p$ drawn i.i.d. from $\mathcal{X}$, we derive an embedding of $f \in \mathcal{H}$ as follows, ensuring that $\langle \tilde{f}, \tilde{h}\rangle \approx \langle f, h\rangle_{\mathcal{H}}$ for all $f, h \in \mathcal{H}$:

$$\tilde{f} := K_W^{-1/2} f_W, \quad \text{where} \quad K_W := (k(\omega_i, \omega_j))_{ij} \in \mathbb{R}^{p\times p}, \quad f_W := (f(\omega_1), \ldots, f(\omega_p))^\top \in \mathbb{R}^p.$$

Let $\tilde{\varphi}(x)$ be the embedding of $k(x, \cdot)$, and $\tilde{Z} := (\tilde{z}_1, \ldots, \tilde{z}_m)$ where $\tilde{z}_i$ is the embedding of the invariance representer z_i. Then [36] showed that the Euclidean embedding of $k^\circ(x, \cdot)$ can be written as

$$(I + \tilde{Z}\tilde{Z}^\top)^{-1/2}\tilde{\varphi}(x). \tag{17}$$

Now to apply proximal map, it is natural to set $L(f) = \frac{1}{2}\sum_{i=1}^m \langle z_i, f\rangle_{\mathcal{H}}^2$ to enforce invariance. Then the proximal map $\mathsf{P}_L(k(x, \cdot))$ for the representer $k(x, \cdot)$ with $\lambda = 1$ is

$$\mathsf{P}_L(k(x, \cdot)) = \arg\min_{f\in\mathcal{H}} \left\{ L(f) + \tfrac{1}{2}\|f - k(x, \cdot)\|_{\mathcal{H}}^2 \right\} \tag{18}$$

$$= \arg\min_{f\in\mathcal{H}} \left\{ \tfrac{1}{2}\sum\nolimits_{i=1}^m \langle z_i, f\rangle_{\mathcal{H}}^2 + \tfrac{1}{2}\|f - k(x, \cdot)\|_{\mathcal{H}}^2 \right\} \tag{19}$$

$$= (I + ZZ^\top)^{-1} k(x, \cdot). \tag{20}$$

Its Euclidean embedding can be obtained by replacing z_i with $\tilde{z}_i$, and $k(x, \cdot)$ with $\tilde{\varphi}(x)$:

$$\arg\min_{v\in\mathbb{R}^p} \left\{ \tfrac{1}{2}\sum\nolimits_{i=1}^m \langle \tilde{z}_i, v\rangle^2 + \tfrac{1}{2}\|v - \tilde{\varphi}(x)\|^2 \right\} = (I + \tilde{Z}\tilde{Z}^\top)^{-1}\tilde{\varphi}(x). \tag{21}$$

This is almost the same as that from kernel warping in (17), except for the exponent on $I + \tilde{Z}\tilde{Z}^\top$. In practice, we observed that it led to little difference, and the result of proximal mapping using Gaussian kernel and flat-gradient invariance is shown in Figure 1. That is, $L(f) = \frac{1}{2}\sum_i \|\nabla f(x_i)\|^2$. Trivially, CKNs can now leverage nonlinear invariances such as total variation by using a nonlinear regularizer L in (18).

D Simulations for Connecting Proximal Mapping to Dropout

We now use the two-moon dataset to verify that only small differences arise if dropout is implemented by proximal mapping in Section 3, as opposed to the adaptive regularization in (6). Suppose the

i-th training examples is $x_i \in \mathbb{R}^d$ with label $y_i \in \{-1, 1\}$. The j-th feature of x_i is denoted as x_{ij}. Employing logistic loss, the adaptive regularization view of dropout by [37] can be written as

$$\beta_* := \min_{\beta \in \mathbb{R}^d} \left\{ \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \beta^\top x_i)) + \mu \sum_j a_j \beta_j^2 \right\}, \tag{22}$$

where $a_j = \frac{1}{n} \sum_{i=1}^{n} p_i(1 - p_i) x_{ij}^2, \quad p_i = (1 + \exp(-\beta^\top x_i))^{-1}$.

Our proximal map is defined as

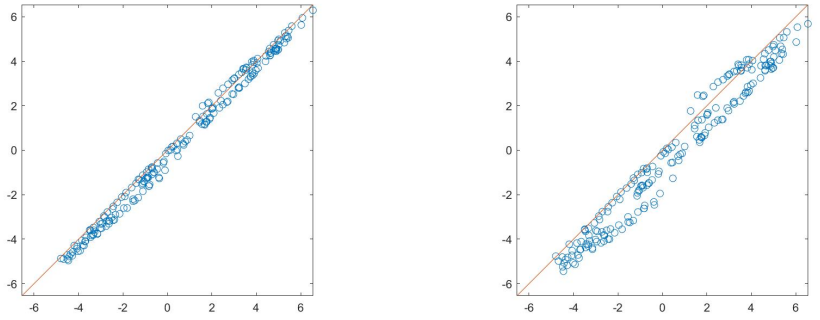
$$\mathsf{P}_R(x) = \arg\min_{z \in \mathbb{R}^d} \left\{ \frac{\lambda}{2} \|z - x\|_2^2 + \sum_j b_j z_j^2 \right\}, \tag{23}$$

where $b_j = \frac{1}{n} \sum_{i=1}^{n} q_i(1 - q_i) x_{ij}^2, \quad q_i = (1 + \exp(-z^\top x_i))^{-1}$.

And the output layer is trained by

$$\alpha_* := \min_{\alpha \in \mathbb{R}^d} \left\{ \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \alpha^\top \mathsf{P}_R(x_i)) + c \|\alpha\|^2 \right\}. \tag{24}$$

To demonstrate that the two methods yield similar discriminant values, we produce a scatter plot of $\alpha_*^\top P_R(x_i)$ (for proximal mapping) versus $\beta_*^\top x_i$ (for dropout). Figure 6 shows the result for two example settings. Clearly, the two methods produce similar discriminant values for all training examples. The Matlab code is also available on GitHub.



(a) $\lambda = 0.5$, $\mu = 0.1$, and $c = 0.2\lambda^2\mu$ (b) $\mu = 0.1$, $\lambda = 0.1$, and $c = 15\lambda^2\mu$

Figure 6: Scatter plot of $\alpha_*^\top P_R(x_i)$ (y-axis for proximal mapping) versus $\beta_*^\top x_i$ (x-axis for dropout)

For optimize (24), we simply invoked fminunc without providing any gradient subroutine. That is, fminunc was98 left to choose its own solver which typically utilizes its own finite difference routine. The result looks good and efficient for this dataset.

E ProxNet for Multiview Learning

Most multiview learning algorithms are based on CCA, which most commonly involves only two views. It is in fact not hard to extend it to more than two views. For example, [78] proposed that given J centered views $X_j \in \mathbb{R}^{N \times d_j}$ for $j \in [J]$, where N is the number of training examples and d_j is the dimensionality of the j-th view, the generalized CCA (GCCA) can be written as the following optimization problem

$$L(\{X_j\}_{j=1}^J) := \min \sum_{j=1}^{J} \|G - X_j U_j\|_F^2, \tag{25}$$

where $G \in \mathbb{R}^{N\times r}$, $U_j \in \mathbb{R}^{d_j\times r}$, $G^\top G = I$. Intuitively, it finds a linear transformation U_j for each view, so that all views can be transformed to a similar core G. Furthermore, G needs to be orthonormal, to avoid mode collapse. The optimal value, denoted as $L(\{X_j\})$, will be used as the L function in (11).

Furthermore, given $\{X_j\}$, (25) can be optimized efficiently in closed form based on generalized eigenvalues [78–80]. Based on the optimal solution of G and $\{U_j\}$, the derivative of $L(\{X_j\})$ in $\{X_j\}$ can be directly computed by Danskin's theorem.

F Backpropagation Through Time for Adversarial LSTM

To concentrate on backpropagation, we assume that the ultimate objective J only depends only on the output of the last time step T, i.e., h_T. Extension can be easily made to the case where each step also contributes to the overall loss. From the final layer, we get $\frac{\partial J}{\partial h_T}$. Then we can get $\frac{\partial J}{\partial h_{T-1}}$ and $\frac{\partial J}{\partial c_{T-1}}$ as in the standard LSTM (G_T in the final layer can be ignored and $\frac{\partial J}{\partial c_T} = 0$). In order to compute the derivatives with respect to the weights W in the LSTMs, we need to recursively compute $\frac{\partial J}{\partial h_{t-1}}$ and $\frac{\partial J}{\partial c_{t-1}}$, given $\frac{\partial J}{\partial h_t}$ and $\frac{\partial J}{\partial c_t}$. Once they are available, then

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \left\{ \underbrace{\frac{\partial J}{\partial h_t}}_{\text{by (27)}} \underbrace{\frac{\partial}{\partial W} h_t(c_{t-1}, h_{t-1}, x_t)}_{\text{standard LSTM}} + \underbrace{\frac{\partial J}{\partial c_t}}_{\text{by (30)}} \underbrace{\frac{\partial}{\partial W} c_t(c_{t-1}, h_{t-1}, x_t)}_{\text{standard LSTM}} \right\}, \tag{26}$$

where the two $\frac{\partial}{\partial W}$ on the right-hand side are identical to the standard operations in LSTMs. Here we use the Jacobian matrix arrangement for partial derivatives, i.e., if f maps from $\mathbb{R}^n$ to $\mathbb{R}^m$, then $\frac{\partial f(x)}{\partial x} \in \mathbb{R}^{m\times n}$.

Given $\frac{\partial J}{\partial c_t}$, we can first compute $\frac{\partial J}{\partial s_t}$ and $\frac{\partial J}{\partial G_t}$ based on the proximal map, and the details will be provided in Section F.1. Given their values, we now compute $\frac{\partial J}{\partial h_{t-1}}$ and $\frac{\partial J}{\partial c_{t-1}}$. Firstly,

$$\frac{\partial J}{\partial h_{t-1}} = \underbrace{\frac{\partial J}{\partial h_t}}_{\text{by recursion}} \underbrace{\frac{\partial h_t}{\partial h_{t-1}}}_{\text{std LSTM}} + \underbrace{\frac{\partial J}{\partial G_t} \frac{\partial G_t}{\partial h_{t-1}}}_{\text{by (28)}} + \underbrace{\frac{\partial J}{\partial s_t}}_{\text{by (37)}} \underbrace{\frac{\partial s_t}{\partial h_{t-1}}}_{\text{std LSTM}}. \tag{27}$$

The terms $\frac{\partial h_t}{\partial h_{t-1}}$ and $\frac{\partial s_t}{\partial h_{t-1}}$ are identical to the operations in the standard LSTM. The only remaining term is in fact a directional second-order derivative, where the direction $\frac{\partial J}{\partial G_t}$ can be computed from from (47):

$$\frac{\partial J}{\partial G_t} \frac{\partial G_t}{\partial h_{t-1}} = \frac{\partial J}{\partial G_t} \frac{\partial^2}{\partial x_t \partial h_{t-1}} s_t(c_{t-1}, h_{t-1}, x_t) \tag{28}$$

$$= \frac{\partial}{\partial h_{t-1}} \left\langle \underbrace{\frac{\partial J}{\partial G_t}}_{\text{by (47)}}, \frac{\partial}{\partial x_t} s_t(c_{t-1}, h_{t-1}, x_t) \right\rangle. \tag{29}$$

Such computations are well supported in most deep learning packages, such as PyTorch. Secondly,

$$\frac{\partial J}{\partial c_{t-1}} = \underbrace{\frac{\partial J}{\partial h_t}}_{\text{by recursion}} \underbrace{\frac{\partial h_t}{\partial c_{t-1}}}_{\text{std LSTM}} + \underbrace{\frac{\partial J}{\partial G_t} \frac{\partial G_t}{\partial c_{t-1}}}_{\text{by (31)}} + \underbrace{\frac{\partial J}{\partial s_t}}_{\text{by (37)}} \underbrace{\frac{\partial s_t}{\partial c_{t-1}}}_{\text{std LSTM}}. \tag{30}$$

The terms $\frac{\partial h_t}{\partial c_{t-1}}$ and $\frac{\partial s_t}{\partial c_{t-1}}$ are identical to the operations in the standard LSTM. The only remaining term is in fact a directional second-order derivative:

$$\frac{\partial J}{\partial G_t} \frac{\partial G_t}{\partial c_{t-1}} = \frac{\partial J}{\partial G_t} \frac{\partial^2}{\partial x_t \partial c_{t-1}} s_t(c_{t-1}, h_{t-1}, x_t) \tag{31}$$

$$= \frac{\partial}{\partial c_{t-1}} \left\langle \underbrace{\frac{\partial J}{\partial G_t}}_{\text{by (47)}}, \frac{\partial}{\partial x_t} s_t(c_{t-1}, h_{t-1}, x_t) \right\rangle. \tag{32}$$

F.1 Gradient Derivation for the Proximal Map

We now compute the derivatives involved in the proximal operator, namely $\frac{\partial J}{\partial s_t}$ and $\frac{\partial J}{\partial G_t}$. For clarify, let us omit the step index t, set $\delta = \sqrt{\lambda}$ without loss of generality, and denote

$$J = f(c), \quad \text{where} \quad c := c(G, s) := (I + GG^\top)^{-1}s. \tag{33}$$

We first compute $\partial J/\partial s$ which is easier.

$$\begin{aligned}\Delta J :=& f(c(G, s + \Delta s)) - f(c(G, s)) && (34)\\ =& \nabla f(c)^\top (c(G, s + \Delta s) - c(G, s)) + o(\|\Delta s\|) && (35)\\ =& \nabla f(c)^\top (I + GG^\top)^{-1}\Delta s + o(\|\Delta s\|). && (36)\end{aligned}$$

Therefore,

$$\frac{\partial J}{\partial s} = \nabla f(c)^\top (I + GG^\top)^{-1}. \tag{37}$$

We now move on to $\partial J/\partial G$. Notice

$$\begin{aligned}\Delta J :=& f(c(G + \Delta G, s)) - f(c(G, s)) && (38)\\ =& \nabla f(c)^\top (c(G + \Delta G, s) - c(G, s)) + o(\|\Delta G\|). && (39)\end{aligned}$$

Since

$$c(G + \Delta G, s) = (I + (G + \Delta G)(G + \Delta G)^\top)^{-1}s \tag{40}$$

$$= \left[(I + GG^\top)^{\frac{1}{2}} \left(I + (I + GG^\top)^{-\frac{1}{2}}(\Delta GG^\top + G\Delta G^\top)(I + GG^\top)^{-\frac{1}{2}}\right)(I + GG^\top)^{\frac{1}{2}}\right]^{-1} s \tag{41}$$

$$= (I + GG^\top)^{-\frac{1}{2}} \left(I - (I + GG^\top)^{-\frac{1}{2}}(\Delta GG^\top + G\Delta G^\top)(I + GG^\top)^{-\frac{1}{2}} + o(\|\Delta G\|)\right)(I + GG^\top)^{-\frac{1}{2}} s \tag{42}$$

$$= c(G, s) - (I + GG^\top)^{-1}(\Delta GG^\top + G\Delta G^\top)(I + GG^\top)^{-1}s + o(\|\Delta G\|), \tag{43}$$

we can finally obtain

$$\begin{aligned}\Delta J &= -\nabla f(c)^\top (I + GG^\top)^{-1}(\Delta GG^\top + G\Delta G^\top)(I + GG^\top)^{-1}s + o(\|\Delta G\|) && (44)\\ &= -\operatorname{tr}\left(\Delta G^\top (I + GG^\top)^{-1}\left(\nabla f(c)s^\top + s\nabla f(c)^\top\right)(I + GG^\top)^{-1}G\right) + o(\|\Delta G\|). && (45)\end{aligned}$$

So in conclusion,

$$\begin{aligned}\frac{\partial J}{\partial G} &= -(I + GG^\top)^{-1}\left(\nabla f(c)s^\top + s\nabla f(c)^\top\right)(I + GG^\top)^{-1}G && (46)\\ &= -(ac^\top + ca^\top)G, \quad \text{where} \quad a = (I + GG^\top)^{-1}\nabla f(c). && (47)\end{aligned}$$

G Detailed Experimental Result

We will demonstrate the effectiveness of ProxNet on several multi-view learning tasks including image classification, speech recognition, and crosslingual word embedding. Four baseline methods were selected for comparison in multi-view learning:

- **Vanilla** model: a network is trained for each view without CCA regularization, and the output of the two views were combined by averaging their logits for supervised tasks. The network is trained in an end-to-end manner.
- **DCCA** [10]: a network is trained to learn a pair of highly-correlated representations for the two views, which are then used for training the subsequent supervised task. The whole model is trained in a disjoint manner.
- **DCCAE** [9]: same as DCCA, except that it trains an extra decoder to enforce that the learned representations can well reconstruct the input.
- **RRM**: connect the code/output of DCCA with a supervised classifier, and train it with the encoder in an end-to-end fashion. It also resembles ProxNet, except that the regularizer $L(X, Y)$ is moved from the proximal layer to the overall objective as in (1) (i.e., no more proximal mapping).

G.1 Multiview Supervised Learning: image recognition with sketch and photo

Dataset. We first evaluated ProxNet on a large scale sketch-photo paired database – Sketchy which consists of 12,500 photos and 75,471 hand-drawn sketches of objects from 125 classes. Each sample from sketch and photo is 256×256 colored natural images. To demonstrate the robustness of our method, we varied the number of classes over $\{20, 50, 100, 125\}$ by sampling a subset of classes from the original dataset. For each class, there are 100 sketch-photo pairs. We randomly selected 80 pairs of photo and sketch from each same class to form the training set, and then used the remaining 20 pairs for testing.

Implementation detail. Our implementation was based on PyTorch and all training was conducted on one NVIDIA GeForce 2080Ti GPU.

For all methods, we used ResNet-18 as the feature extractor. In ProxNet, the feature extractor immediately followed by a proximal layer which has input and output dimension $d = 20$. Then a classifier which has three fully-connected layer each having 512 units was trained on the outputs of proximal layer. The final output layer has multiple softmax units that each corresponds to the output classes. At training time, we employed an adaptive trade-off parameter $\lambda_t = (1 + kt)\alpha_0$, where $k = 0.5$ and $\alpha_0 = 0.1$. RRM used the same architecture as ProxNet, except that, instead of using the proximal layer, RRM moves the CCA objective (i.e., the regularizer $L(X, Y)$) to the overall objective to promote the correlation between the two views' hidden representation.

Since the Vanilla model does not promote correlation between views, it can be adapted from RRM model by removing the regularizer from the overall objective. [9, 10] trained DCCA and DCCAE in two separate steps instead of end-to-end. The first step learned an encoder (and decoder for DCCAE) to optimize the CCA objective, and the second step trained a supervised classifier based on the code. In our experiment, their encoders employed the same architecture as the feature extractors of ProxNet and other baselines, i.e., ResNet-18. For DCCAE, we built a CNN-based decoder to reconstruct the inputs.

For all methods, the loss was evaluated on the averaged logits at training time in order to be consistent with how predictions were made at test time.

We used the Ray Tune library to select the hyper-parameters for all methods, and the selected parameters are summarized here:

Table 5: Hyper-parameters for all methods on the Sketchy dataset

Hyper-parameters	Vanilla	DCCA	DCCAE	RRM	ProxNet
Dimension d	15	22	19	20	20
Optimizer	Adam	Adam	Adam	Adam	Adam
Learning rate	0.0012	0.0010	0.0009	0.0011	0.0011
Weight decay	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}

The accuracy of all methods saturates after the mini-batch size goes above 100. So we just used 100 for all methods to keep training efficient.

G.2 Audio-Visual Speech Recognition

Dataset. In this task, we aim to use learned features for speaker-independent phonetic recognition. We experimented on the Wisconsin X-ray Micro-Beam Database (XRMB) corpus which consists of simultaneously recorded speech and articulatory measurements from 47 American English speakers and 2357 utterances. The two raw-input views are acoustic features (39D mel frequency cepstral coefficients (MFCCs) and their first and sencond derivatives) and articulatory features (16D horizontal/vertical displacement of 8 pellets attached to several parts of the vocal tract). Along with the multi-view data there are phonetic labels available for classification. To simulate the real-life scenarios and improve the model's robustness to noise, the acoustic features of a given speaker are corrupted by mixing with $\{0.2, 0.5, 0.8\}$ level of another random picked speaker's acoustic features. The XRMB speakers were partitioned into disjoint sets of 35/12 speakers for training and testing respectively.

Implementation detail. In [9], to incorporate contexts information, the inputs are concatenated over a W-frame window centered at each frame, giving $39 \times W$ and $16 \times W$ feature dimensions for each of the two views respectively. Although this delicately construed inputs freed the encoder/feature extractor from considering the time dependency within frames, we prefer a refined modeling of the sequential structure. Therefore, instead of concatenating features for each W-frame window followed by a fully connected network as in [45], we implementated, for all algorithms under consideration, a 2-layer LSTM with hidden size 256. The output of LSTM was passed through a fully connected layer, projecting to a K-dimensional subspace. This feature extractor significantly improved the performance of all methods.

The supervised predictor was implemented by a fully connected network of 2 hidden layers each having 256 ReLU units, and a linear output layer of 41 log-softmax units. We used Pytorch's built-in function Connectionist Temporal Classification (CTC) loss [44] with greedy search as the phone recognizer. Again, all methods shared the same architecture of supervised predictor.

Both RRM and Vanilla were trained in the same way as for the Sketchy dataset in Section G.1. To train ProxNet, we employed an adaptive trade-off parameter $\lambda_t = (1 + kt)\alpha_0$, where $k = 1$ and $\alpha_0 = 0.5$. DCCA and DCCAE performed poorly if only the learned code/features were used for phonetic recognition. Therefore, we followed [9] and concatenated them with the original features (39D and 16D for the acoustic and articulatory views, respectively), based on which a CTC-based recognizer is trained. This improved the PER performance of DCCA and DCCAE significantly.

In the logit averaging mode, all methods were trained with a loss applied to the averaged logits. This is the same as Section G.1. In the acoustic mode, however, a loss is applied to each view at training time based on the ground truth label. These are both consistent with how predictions are made at test time.

Here we intentionally used K instead of d to denote the hidden dimension. This is to avoid confusion because LSTM is used as in Figure 4. For a mini-batch of size m where each sequence has length s, the input of the proximal layer is in fact $m \cdot s$ examples of K dimensional. Although $m \cdot s$ may result in a large number, the proximal mapping can still be solved efficiently because we were able to use a larger value of λ in this dataset. In addition, the computational cost for SVD on an ms-by-K matrix is $O(msK^2)$ when $K \leq ms$. Since we used $K = 20$, the quadratic dependency on K did not create a computational challenge in practice.

As in the Skytch dataset, we used the Ray Tune library to select the hyper-parameters, and the selected parameters are summarized here:

Table 6: Hyper-parameters for all methods on XRMB

Hyper-parameters	Vanilla	DCCA	DCCAE	RRM	ProxNet
Dimension K	12	20	20	18	20
Optimizer	Adam	Adam	Adam	Adam	Adam
Learning rate	0.0009	0.0011	0.0010	0.0013	0.0010
Weight decay	0.0005	0.0005	0.0005	0.0005	0.0005

Line 250 made an inaccurate description of how we tuned K: "The dimension of subspace was tuned in $\{10, 20, 30, 50\}$, and the sequence length was tuned in $\{250, 500, 1000\}$ for all algorithms". This was the setting in our preliminary experiment. The Ray Tune library indeed allowed us to later search all parameters in a continuous space, and so the K values in Table 6 can be 12 or 18.

We eventually set the sequence length to $s = 1000$ for all methods, because it consistently produced the best result, which is not surprising because longer sequences can preserve more structure. However, the PER saturated after the length rose beyond 1000.

Similarly, the PER of all methods leveled off after the mini-batch size grew above 32. So we just used $m = 32$ for all methods to keep training efficient.

Evaluation. For all experiments, we report the Phone error rates (PERs) which is defined as $PER = (S + D + I)/N$, where S is the number of substitutions, D is the number of deletions,

I is the number of insertions to get from the reference to the hypothesis, and N is the number of phonetics in the reference. The PERs obtained by different methods are given in Table 2.

G.3 Crosslingual/Multilingual Word Embedding

In this task, we learned representation of English and German words from the paired (English, German) word embeddings for improved semantic similarity.

Dataset. We first built a parallel vocabulary of English and German from the parallel news commentary corpora [WMT 2012-2018 46] using the word alignment method from [47, 48]. Then we selected 36K English-German word pairs, in descending order of frequency, for training. Based on the vocabulary we also built a bilingual dictionary for testing, where each English word x_i is matched with the (unique) German word y_i that has been most frequently aligned to x_i. Unlike the setup in [53] and [9], where word embeddings are trained via Latent Semantic Analysis (LSA) using parallel corpora, we used the pretrained monolingual 300-dimensional word embedding from [50] and [49] as the raw word embeddings (x_i and y_i).

To evaluate the quality of learned word representation, we experimented on two different benchmarks that have been widely used to measure word similarity [51, 52]. Multilingual WS353 contains 353 pairs of English words, and their translations to German, Italian and Russian, that have been assigned similarity ratings by humans. It was further split into Multilingual WS-SIM and Multilingual WS-REL which measure the similarity and relatedness between word pairs respectively. Multilingual SimLex999 is a similarity-focused dataset consisting of 666 noun pairs, 222 verb pairs, 111 adjective pairs, and their translations from English to German, Italian and Russian.

Baselines. We compared our method with the monolingual word embedding (baseline method) from fastText to show that ProxNet learned a good word representation through the proximal layer. Since our method is mainly based on CCA, we also chose three competitive CCA-based models for comparison, including:

- linearCCA [53], which applied a linear projection on the two languages' word embedding and then projected them into a common vector space such that aligned word pairs should be maximally correlated.
- DCCA [81], which, instead of learning linear transformations with CCA, learned nonlinear transformations of two languages' embedding that are highly correlated.
- DCCAE [9], which noted that there is useful information in the original inputs that is not correlated across views. Therefore, they not only projected the original embedding into subspace, but also reconstructed the inputs from the latent representation.
- CL-DEPEMB [54], a novel cross-lingual word representation model which injects syntactic information through dependency-based contexts into a shared cross-lingual word vector space.

Implementation detail. We first used the fastText model to embed the 36K English-German word pairs into vectors. Then we normalized each vector to unit ℓ_2 norm and removed the per-dimension mean and standard deviation of the training pairs.

To build an end-to-end model, we followed the same intuition as DCCAE but instead of using the latent representation from the encoder to reconstruct the inputs, we used the outputs of proximal layer, which is a proximal approximation of latent representation from the encoder, to do the reconstruction. That is, the input reconstruction error was used as the ultimate objective.

We implemented the encoder (feature mapping f and g) by using multilayer perceptrons with ReLU activation and the decoder by using a symmetric architecture of encoder. We tuned the hidden dimension h for f and g among $\{0.1, 0.3, 0.5, 0.7, 0.9\} \times 300$, the regularization parameter λ from $\{0.001, 0.01, 0.1, 1, 10\}$, and the depth and layer width from 1 to 4 and $\{256, 512, 1024, 2048\}$, respectively. For optimization, we used SGD with momentum 0.99, a weight decay of 0.0005, and a learning rate 0.1 which was divided by 10 after 100 and 200 epochs.

At test time, for numerical stability, we combined the word vectors from bilingual dictionary and the test set to build paired vocabulary for each language. We applied the same data preprocessing

Table 7: Summary of datasets for adversarial LSTM training

Dataset	Training	Test	Median length	Attributes	Classes
JV	225	370	15	12	9
HAR	6,127	2,974	128	9	6
AD	5,500	2,200	39	13	10
IMDB	25,000	25,000	239	-	2

(normalize to unit norm, remove the mean/standard deviation of the training set) on test vocabularies (English and German word vectors). Then we feed paired test vocabularies into the models and obtained the word representation of the test data. We projected the output of the proximal layer to the subspace where each paired word representation was maximally correlated. The projection matrices were calculated from the 36K training set through the standard CCA method. We computed the cosine similarity between the final word vectors in each pair, ordered the pairs by similarity, and computed the Spearman's correlation between the model's ranking and human's ranking.

G.4 Adversarial Training in Recurrent Neural Network

Here we include more details on the experiment of adversarial training in recurrent neural network as described in Section 6.4.

Datasets. To demonstrate the effectiveness of using proximal mapping, we tested on four different sequence datasets. The Janpanese Vowels dataset [JV 55] contains time series data where nine male speakers uttered Japanese Vowels successively, and the task is to classify speakers. The Human Activity Recognition dataset [HAR 56] is used to classify a person's activity (sitting, walking, etc.) based on a trace of their movement using sensors. The Arabic Digits dataset [AD, 57] contains time series corresponding to spoken Arabic digits by native speakers, and the task is to classify digits. IMDB [58] is a standard movie review dataset for sentiment classification. Details of the datasets are summarized in Table 7. The - is because IMDB is a text dataset, for which a 256-dimensional word embedding is learned.

Preprocessing. Normalization was the only preprocessing applied to all datasets. For those datasets that contain variable-length sequences, zero-padding was used to make all sequences have the same length as the longest sequence in a mini-batch. To reduce the effect of padding, we first sorted all sequences by length (except the IMDB dataset), so that sequences with similar length were assigned to the same mini-batch.

Baseline models. To show the impact of applying proximal mapping on LSTM, we compared our method with two baselines. For JV, HAR and AD datasets, the base model structure was composed of a CNN layer, followed by an LSTM layer and a fully-connected layer. The CNN layer was constructed with kernel size 3, 8, 3 and contained 32, 64, 64 filters for JV, HAR, AD respectively. For the LSTM layer, the number of hidden units used in these three datasets are 64, 128, 64, respectively. This architecture was denoted as LSTM in Table 4. For IMDB, following [40], the basic model consisted of a word embedding layer with dimension 256, a single-layer LSTM with 1024 hidden units, and a hidden dense layer of dimension 30.

On top of this basic LSTM structure, we compared two different adversarial training methods. AdvLSTM is the adversarial training method in [40], which we reimplemented in PyTorch, and perturbation was added to the input of each LSTM layer. ProxLSTM denotes our method described in Section 4, where the LSTM cell in the basic structure was replaced by our ProxLSTM cell. LSTM and AdvLSTM here correspond to "Baseline" and "Adversarial" in [40] respectively.

Training. For the JV, HAR, AD datasets, we first trained the baseline LSTM to convergence, and then applied AdvLSTM and ProxLSTM as fine tunning, where ADAM was used with learning rate 10^{-3} and weight decay 10^{-4}. For IMDB, we first trained LSTM and AdvLSTM by following the settings in [40], with an ADAM optimizer of learning rate $5 \cdot 10^{-4}$ and exponential decay 0.9998. Then the result of AdvLSTM was used to initialize the weights of ProxLSTM. All settings were evaluated 10 times to report the mean and standard deviation.

Results. The test accuracies were summarized in Table 4. Clearly, adversarial training improves the performance, and ProxLSTM even promotes the performance more than AdvLSTM. Figure 5 illustrates the t-SNE embedding of extracted features from the last time step's hidden state of HAR test set. Although ProxLSTM only improves upon AdvLSTM marginally in test accuracy, Figure 5 shows the embedded features from ProxLSTM cluster more compactly than those of AdvLSTM (e.g. the yellow class). The t-SNE plot of other datasets are available in Figures 7, 8 and 9. This further indicates that ProxLSTM can learn better latent representation than AdvLSTM by applying proximal mapping.

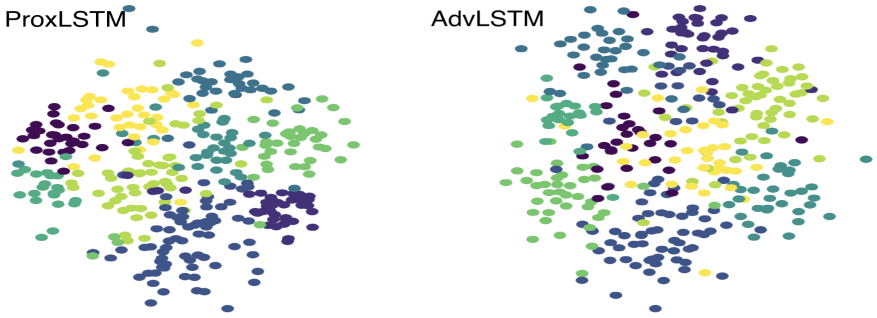


Figure 7: t-SNE embedding of the JV dataset

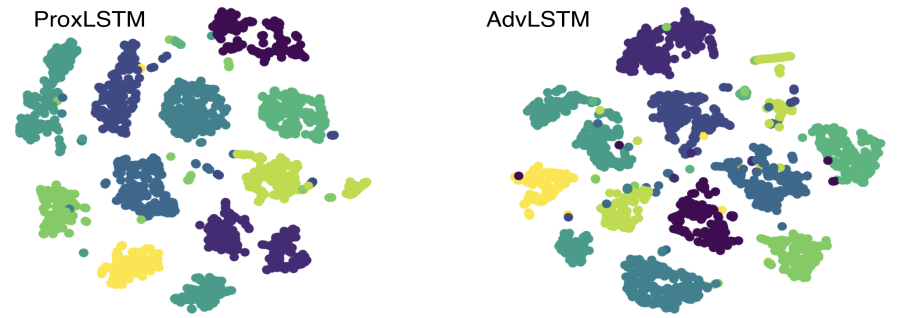


Figure 8: t-SNE embedding of the AD dataset

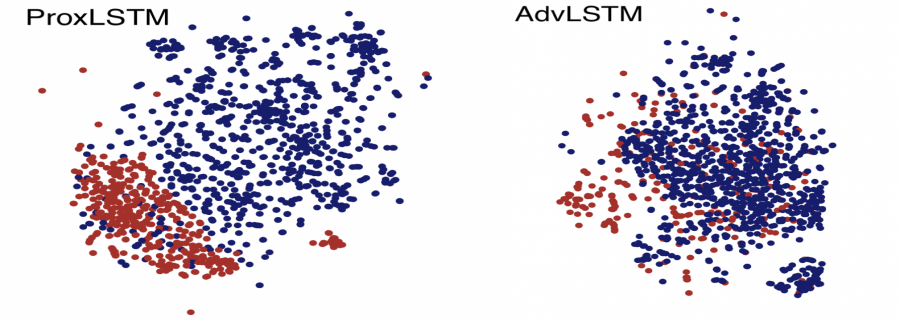


Figure 9: t-SNE embedding of the IMDB dataset