Learning Optimal Linear Regularizers

Matthew Streeter [1]

Abstract

We present algorithms for efficiently learning regularizers that improve generalization. Our approach is based on the insight that regularizers can be viewed as upper bounds on the generalization gap, and that reducing the slack in the bound can improve performance on test data. For a broad class of regularizers, the hyperparameters that give the best upper bound can be computed using linear programming. Under certain Bayesian assumptions, solving the LP lets us "jump" to the optimal hyperparameters given very limited data. This suggests a natural algorithm for tuning regularization hyperparameters, which we show to be effective on both real and synthetic data.

1. Introduction

Most machine learning models are obtained by minimizing a loss function, but optimizing the training loss is rarely the ultimate goal. Instead, the model is ultimately judged based on information that is unavailable during training, such as performance on held-out test data. The ultimate value of a model therefore depends critically on the loss function one chooses to minimize.

Traditional loss functions used in statistical learning are the sum of two terms: the empirical training loss and a regularization penalty. A common regularization penalty is the ℓ^1 or ℓ^2 norm of the model parameters. More recently, it has become common to regularize implicitly by perturbing the examples, as in dropout (Srivastava et al., 2014), or perturbing the labels, as in label smoothing (Szegedy et al., 2016), or by modifying the training algorithm, as in early stopping (Caruana et al., 2001).

The best choice of regularizer is usually not obvious *a priori*. Typically one chooses a regularizer that has worked well on similar problems, and then fine-tunes it by searching for the hyperparameter values that give the best performance on a held-out validation set. Though this approach can be effective, it tends to require a large number of training runs, and to mitigate this the number of hyperparameters must be kept fairly small in practice.

In this work, we seek to recast the problem of choosing regularization hyperparameters as a supervised learning problem which can be solved more efficiently than is possible with a purely black-box approach. Specifically, we show that the optimal regularizer is by definition the one that provides the tightest possible bound on the generalization gap (i.e., difference between test and training loss), for a suitable notion of "tightness". We then present an algorithm that can find approximately optimal regularization hyperparameters efficiently via linear programming. Our method applies to explicit regularizers such as L2, but also in an approximate way to implicit regularizers such as dropout.

Our algorithm takes as input a small set of models for which we have computed both training and validation loss, and produces a set of recommended regularization hyperparameters as output. Under certain Bayesian assumptions, we show that our algorithm returns the optimal regularization hyperparameters, requiring data from as few as *two* training runs as input. Building on this linear programming algorithm, we present a hyperparameter tuning algorithm and show that it outperforms state-of-the-art alternatives on real problems where these assumptions do not hold.

1.1. Definitions and Notation

We consider a general learning problem with an arbitrary loss function. Our goal is to choose a model θ from a set Θ, so as to minimize the expected value of a non-negative loss $\ell(z, \theta)$, where z is an example drawn from an unknown distribution $\mathcal{D}$. That is, we wish to minimize

$$L(\theta) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(z, \theta)] \, .$$

In a typical supervised learning problem, θ is a parameter vector, z is a (feature vector, label) pair, and ℓ is a loss function such as log loss or squared error. In an unsupervised problem, z might be an unlabeled image or a text fragment.

We assume as input a set $\{z_i \mid 1 \leq i \leq n\}$ of training examples. Where noted, we assume each z_i is sampled indepen-

[1]Google Research. Correspondence to: Matthew Streeter <mstreeter@google.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

dently from $\mathcal{D}$. We denote average training loss by:

$$\hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(z_i, \theta) \; .$$

We will focus on algorithms that minimize an objective function $f(\theta) = \hat{L}(\theta) + R(\theta)$, where $R : \Theta \to \mathbb{R}_{>0}$ is the regularizer (the choice of which may depend on the training examples). We denote the minimizer of regularized loss by

$$\hat{\theta} = \operatorname*{argmin}_{\theta \in \Theta} \left\{ \hat{L}(\theta) + R(\theta) \right\}$$

and the optimal model by

$$\theta^* = \operatorname*{argmin}_{\theta \in \Theta} \left\{ L(\theta) \right\} \; .$$

We refer to the gap $L(\hat{\theta}) - L(\theta^*)$ as *excess test loss*. Our goal is to choose R so that the excess test loss is as small as possible.

2. Regularizers and Generalization Bounds

Our strategy for learning regularizers will be to compute a regularizer that provides the tightest possible estimate of the generalization gap (difference between test and training loss). To explain the approach, we begin with a mathematically trivial yet surprisingly useful observation:

$$R(\theta) = L(\theta) - \hat{L}(\theta) \implies L(\hat{\theta}) = L(\theta^*) \; .$$

That is, the generalization gap is by definition an optimal regularizer, since training with this regularizer amounts to training directly on the test loss. More generally, for any monotone function h, the regularizer $R(\theta) = h(L(\theta)) - \hat{L}(\theta)$ is optimal.

Though training directly on the test loss is clearly not a workable strategy, we might hope to use a regularizer that accurately estimates the generalization gap, so that regularized training loss estimates test loss. What makes a good estimate in this context?

In supervised learning we typically seek an estimate with good average-case performance, for example low mean squared error. However, that fact that $\hat{\theta}$ is obtained by optimizing over all $\theta \in \Theta$ means that a single bad estimate could make $L(\hat{\theta})$ arbitrarily bad, suggesting that worst-case error matters. At the same time, it should be less important to estimate $L(\theta)$ accurately if θ is far from optimal. The correct characterization turns out to be:

> *A good regularizer is one that provides an upper bound on the generalization gap that is tight at near-optimal points.*

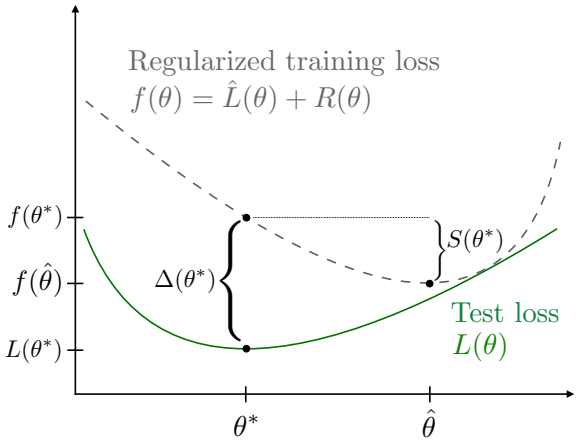

Figure 1. Suboptimality (S) and slack (Δ).

To formalize this, we introduce two quantities. For a fixed regularizer R, defining a regularized training loss $f(\theta) = \hat{L}(\theta) + R(\theta)$, define the *slack* of R at a point θ as

$$\Delta(\theta) \equiv f(\theta) - L(\theta) \; .$$

For any θ, define the *suboptimality* as $S(\theta) \equiv f(\theta) - f(\hat{\theta})$. Figure 1 illustrates these definitions.

We now give an expression for excess test loss in terms of slack and suboptimality. For any θ, $L(\theta) = f(\theta) - \Delta(\theta)$, so

$$\begin{aligned} L(\hat{\theta}) - L(\theta) &= f(\hat{\theta}) - \Delta(\hat{\theta}) - f(\theta) + \Delta(\theta) \\ &= \Delta(\theta) - \Delta(\hat{\theta}) - S(\theta) \\ &\equiv \mathsf{SAS}(\theta) \; . \end{aligned}$$

We refer to the quantity $\mathsf{SAS}(\theta)$ as *suboptimality-adjusted slack*. Maximizing both sides over all $\theta \in \Theta$ gives an expression for excess test loss:

$$L(\hat{\theta}) - L(\theta^*) = \max_{\theta \in \Theta} \left\{ \mathsf{SAS}(\theta) \right\} \; .$$

That is, the excess test loss of a model $\hat{\theta}$ obtained by minimizing $\hat{L}(\theta) + R(\theta)$ is the worst-case suboptimality-adjusted slack. An optimal regularizer is therefore one that minimizes this quantity. This is summarized in the following proposition.

Proposition 1. *For any set Θ of models, and any set $\mathcal{R}$ of regularizers, the optimal regularizer is*

$$\operatorname*{argmin}_{R \in \mathcal{R}} \left\{ L(\hat{\theta}(R)) \right\} = \operatorname*{argmin}_{R \in \mathcal{R}} \left\{ \max_{\theta \in \Theta} \left\{ \mathsf{SAS}(\theta; R) \right\} \right\}$$

where $\hat{\theta}$ and SAS are defined as above, with the dependence on R now made explicit.

How can we make use of Proposition 1 in practice? We do not of course know the test loss for all $\theta \in \Theta$, and thus we cannot compute $\max_{\theta \in \Theta} \{\mathsf{SAS}(\theta; R)\}$ exactly. However, it is feasible to compute the validation loss for a small set $\Theta_0 \subseteq \Theta$ of models, for example by doing multiple training runs with different regularization hyperparameters, or doing a single run with different thresholds for early stopping. We can then compute an approximately optimal regularizer using the approximation:

$$R^* \approx \operatorname*{argmin}_{R \in \mathcal{R}} \left\{ \max_{\theta \in \Theta_0} \left\{ \hat{\mathsf{SAS}}(\theta; R) \right\} \right\} \tag{1}$$

where $\hat{\mathsf{SAS}}$ is an estimate of SAS that uses validation loss as a proxy for test loss, and uses Θ_0 as a proxy for Θ.

Importantly, the R^* given by equation 1 will generally *not* be one of the regularizers we already tried when producing the models in Θ_0, as is shown formally in Theorem 1.

This suggests a simple iterative procedure for tuning regularization hyperparameters. Initially, let Θ_0 be a small set of models trained using a few different hyperparameter settings. Then, use approximation (1) to compute an approximately optimal regularizer R_1 based on Θ_0. Then, train a model θ_1 using R_1, and add it to Θ_0 to obtain a new set $\Theta_1 = \Theta_0 \cup \{\theta_1\}$, use Θ_1 to obtain a better approximation R_2, and so on.

2.1. Implicit Regularizers

So far we have assumed the regularizer is an explicit function of the model parameters, but many regularizers used in deep learning do not take this form. Instead, they operate implicitly by perturbing the weights, labels, or examples. Is Proposition 1 still useful in this case?

It turns out to be straightforward to accomodate such regularizers. To illustrate, let $P(\theta)$ be a (possibly randomized) perturbation applied to θ. Training with the loss function $\mathbb{E}[\hat{L}(P(\theta))]$ is equivalent to using the regularizer

$$R(\theta) = \mathbb{E}[\hat{L}(P(\theta))] - \hat{L}(\theta) \, .$$

In the case of dropout, $P(\theta)$ sets each activation in a layer to 0 independently with some probability p, which is equivalent to setting the corresponding outgoing weights to zero. The regularizer is simply the expected difference between training loss using the perturbed weights and training loss using the original weights. This observation, together with Proposition 1, yields the following conclusion:

> *The best dropout probability is the one that makes the gap between training loss with and without dropout be the tightest possible estimate of the generalization gap (where tightness is worst-case suboptimality-adjusted slack).*

Similar constructions can be used to accomodate implicit regularizers that perturb the labels (as in label smoothing) or the input data (as in data augmentation). More generally, we can view any perturbation as a potentially useful regularizer. For example, training with quantized model weights can be viewed as using a regularizer equal to the increase in training loss that quantization introduces. Quantization will be effective as a regularizer to the extent that this increase provides a tight estimate of the generalization gap.

3. Learning Linear Regularizers

We now consider the problem of selecting a regularizer from a set $\mathcal{R}$ of possible regularizers, given as input a set Θ_0 of models whose training and validation loss are known. In practice, the models in Θ_0 might be the result of training for different amounts of time, or using different hyperparameters.

We consider the case where $\mathcal{R}$ is the set of all regularizers that are linear functions of a user-provided feature vector.

Definition 1. *Let $q : \Theta \to \mathbb{R}^k_{\geq 0}$ be a function that, given a model, returns a feature vector of length k. A regularizer is* linear *with respect to q if, for $\lambda \in \mathbb{R}^k_{\geq 0}$, it can be written as*

$$R(\theta; \lambda) = \lambda \cdot q(\theta) \, .$$

Commonly-used regularizers such as L1 and L2 can be expressed in this form by including the model's L1 or squared L2 norm in the feature vector, and novel regularizers can be easily defined by including additional features. We consider the case where $\mathcal{R} = \left\{ R(\cdot; \lambda) : \lambda \in \mathbb{R}^k_{\geq 0} \right\}$ for some fixed, user-provided feature vector $q(\theta)$.

Dropout is not a linear regularizer, because the implicit regularization penalty, $R(\theta; p) = \mathbb{E}[\hat{L}(\text{dropout}(\theta; p))] - \hat{L}(\theta)$, varies nonlinearly as a function of the dropout probability p. However, dropout can be approximated by a linear regularizer using a feature vector of the form $q(\theta) = \langle R(\theta; p_1), R(\theta; p_2), \ldots, R(\theta; p_k) \rangle$, for a suitably fine grid of dropout probabilties $\{p_1, p_2, \ldots, p_k\}$.

Our algorithm for selecting a linear regularizer is designed to have two desirable properties:

1. *Consistency:* in the limiting case where $\Theta_0 = \Theta$, and validation loss is an exact estimate of test loss, it recovers an optimal regularizer.
2. *Efficiency:* in the case where there exists a regularizer that perfectly estimates the generalization gap, we require only $k + 1$ data points in order to recover it.

To describe our algorithm, let $V(\theta)$ be average loss on a validation set. To guarantee consistency, it is sufficient that we return the regularizer $R \in \mathcal{R}$ that minimizes the validation

loss of $\hat{\theta}_0(R)$, where $\hat{\theta}_0(R)$ is the model in Θ_0 that minimizes regularized training loss when R is the regularizer. That is, we wish to find the regularizer

$$\hat{R} \equiv \operatorname*{argmin}_{R \in \mathcal{R}} \left\{ V(\hat{\theta}_0(R)) \right\}$$

where $\hat{\theta}_0(R) \equiv \operatorname{argmin}_{\theta \in \Theta_0} \left\{ \hat{L}(\theta) + R(\theta) \right\}$.

This regularizer can be computed as follows. For each $\theta_i \in \Theta_0$, we solve a linear program to compute a function of the form $f(\theta) = \hat{L}(\theta) + \lambda \cdot q(\theta)$, subject to the constraint that $\theta_i = \operatorname{argmin}_{\theta \in \Theta_0} \{f(\theta)\}$, or equivalently $f(\theta_i) \leq f(\theta)\ \forall \theta \in \Theta_0$. Among the θ_i's for which the LP is feasible, the one with minimum $V(\theta_i)$ determines $\hat{R}$. Knowing this, we consider the θ_i's in ascending order of $V(\theta_i)$, stopping as soon as we find an LP that is feasible.

To guarantee efficiency, we must break ties appropriately in the case where there are multiple values of λ that produce the same argmin of f. To do this, we first require that f upper bounds $\alpha V(\theta)$ for some learned $\alpha \geq 0$. Note that because f is non-negative, this constraint cannot make the LP infeasible. It then suffices to minimize the total slack in this upper bound, as shown in the proof of Theorem 1.

Algorithm LearnLinReg

Input: Set of (validation loss, training loss, feature vector) tuples $\left\{ (V_i, \hat{L}_i, q_i) \mid 1 \leq i \leq m \right\}$.
Sort tuples in ascending order of V_i, and reindex so that $V_1 \leq V_2 \leq \ldots \leq V_m$.
for i^* from 1 to m **do**
Solve the following linear program:

$$\begin{array}{lll} \text{minimize}_{\lambda \in \mathbb{R}^k_{\geq 0}} & \sum_{i=1}^m \Delta_i & \\ \text{subject to} & \alpha \geq 0 & \\ & \Delta_i \geq 0 & \forall i \\ & f_i = \alpha V_i + \Delta_i & \forall i \\ & f_i = \lambda \cdot q_i + \hat{L}_i & \forall i \\ & f_{i^*} \leq f_i & \forall i \end{array}$$

If the LP is feasible, return (α, λ).
Return error.

By construction, LearnLinReg returns the linear regularizer that would have given the best possible validation loss, when minimizing regularized training loss over all $\theta \in \Theta_0$. This guarantees consistency, as summarized in Proposition 2.

Proposition 2. *Assuming it terminates successfully, LearnLinReg returns a pair (α^*, λ^*) such that*

$$R(\cdot; \lambda^*) = \operatorname*{argmin}_{R \in \mathcal{R}} \left\{ V(\hat{\theta}_0(R)) \right\} .$$

We now consider efficiency. In the case where there exists a λ^* that allows for perfect estimation of validation loss, Theorem 1 shows that LearnLinReg can recover λ^* provided Θ_0 contains as few as $k+1$ models, where k is the size of the feature vector.

Theorem 1. *Suppose there exists a perfect regularizer, in the sense that for some vector λ^* and scalar $\alpha^* > 0$,*

$$\alpha^* V(\theta) = \hat{L}(\theta) + \lambda^* \cdot q(\theta) \qquad \forall \theta \in \Theta \,.$$

Let $D = \left\{ (V(\theta_i), \hat{L}(\theta_i), q(\theta_i)) \mid 1 \leq i \leq k+1 \right\}$ be a set of tuples such that the vectors $\langle V(\theta_i) \rangle \frown q(\theta_i)$ are linearly independent, where $k = |q(\theta)|$. Then, $LearnLinReg(D)$ will return (α^*, λ^*).

Proof. Under these assumptions, the first LP considered by LearnLinReg has a feasible point $\alpha = \alpha^*$, $\lambda = \lambda^*$, and $\Delta_i = 0\ \forall i$. Because each Δ_i is constrained to be non-negative, this point must be optimal, and thus any optimal point must have $\Delta_i = 0\ \forall i$. Thus, any optimal point must satisfy $\alpha V(\theta_i) = \lambda \cdot q(\theta_i) + \hat{L}(\theta_i)\ \forall i$. This is a system of linear equations with $k+1$ variables, and by assumption the equations are linearly independent. Thus the solution is unique, and the algorithm returns (α^*, λ^*). □

If desired, LearnLinReg can be easily modified to only consider λ vectors in a feasible set $\mathcal{F} = \prod_{j=1}^k [\lambda_j^{\min}, \lambda_j^{\max}]$.

3.1. Hyperparameter Tuning

Algorithm TuneReg

Input: training loss $\hat{L}(\theta)$, validation loss $V(\theta)$, feature vector $q(\theta)$, initial set Λ_0 of hyperparameter vectors, training algorithm $\mathrm{train}(\lambda)$, hyperparameter vector sampler $\mathrm{random\_sample}()$.
Set $\Theta_0 \leftarrow \{\mathrm{train}(\lambda) \mid \lambda \in \Lambda_0\}$, $\Lambda_{\text{seen}} \leftarrow \Lambda_0$.
Set $D \leftarrow \left\{ (V(\theta), \hat{L}(\theta), q(\theta)) \mid \theta \in \Theta_0 \right\}$.
while True **do**
Set $\_, \lambda \leftarrow \mathrm{LearnLinReg}(D)$.
If $\lambda \in \Lambda_{\text{seen}}$, set $\lambda \leftarrow \mathrm{random\_sample}()$.
Set $\theta \leftarrow \mathrm{train}(\lambda)$, and set $\Lambda_{\text{seen}} \leftarrow \Lambda \cup \{\lambda\}$
Set $D \leftarrow D \cup \left\{ (V(\theta), \hat{L}(\theta), q(\theta)) \right\}$.

We now describe how to use LearnLinReg for hyperparameter tuning. Given an initial set of hyperparameter vectors, we train using each one, and observe the resulting training and validation loss, as well as the feature vector for the trained model. We then feed this data to LearnLinReg to obtain a vector λ of regularization hyperparameters. We then train using these hyperparameters, add the results to our dataset, and re-run LearnLinReg. Experiments using this algorithm are presented in §5.

4. Recovering Bayes-Optimal Regularizers

We have shown that a regularizer is optimal if it can be used to perfectly predict the generalization gap, and have presented an algorithm that can efficiently recover a *linear* regularizer if a perfect one exists. Do perfect linear regularizers ever exist? Perhaps surprisingly, the answer is "yes" for a broad class of Bayesian inference problems.

As discussed in §1.1, we assume examples are drawn from a distribution $\mathcal{D}$. In the Bayesian setting, we assume that $\mathcal{D}$ itself is drawn from a (known) prior distribution $\mathcal{D}_1$. Given a training dataset $Z \sim \mathcal{D}^n$, where $\mathcal{D} \sim \mathcal{D}_1$, we now care about the conditional expected test loss:

$$\bar{L}(\theta, Z) \equiv E_{\mathcal{D} \sim \mathcal{D}_1}[L(\theta, \mathcal{D})|Z]$$

where $L(\theta, \mathcal{D}) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(z, \theta)]$ as in §1.1. A Bayes-optimal regularizer is one which minimizes this quantity.

Definition 2. *Given a training set $Z \sim \mathcal{D}^n$, where $\mathcal{D} \sim \mathcal{D}_1$, a Bayes-optimal regularizer is a regularizer R^* that satisfies:*

$$\operatorname*{argmin}_{\theta \in \Theta} \left\{ \hat{L}(\theta) + R^*(\theta) \right\} = \operatorname*{argmin}_{\theta \in \Theta} \left\{ \bar{L}(\theta, Z) \right\} .$$

R^* is **perfect** *if, additionally, it satisfies the stronger condition*

$$\hat{L}(\theta) + R^*(\theta) = h(\bar{L}(\theta, Z)) \; \forall \theta \in \Theta$$

for some monotone function h.

Theorem 2 shows that a perfect Bayes-optimal regularizer exists for density estimation problems where log loss is the loss function, θ parameterizes an exponential family distribution $\mathcal{D}$ from which examples are drawn, and $\mathcal{D}_1$ is the conjugate prior for $\mathcal{D}$.

Theorem 2. *Let $P(z|\theta)$ be an exponential family distribution with natural parameter $\eta(\theta)$ and conjugate prior $P(\theta)$, and suppose the following hold:*

1. *$z \sim \mathcal{D}$, where $\mathcal{D} = P(z|\theta^*)$.*
2. *$\ell(z, \theta) = -\log P(z|\theta)$.*
3. *$\theta^* \sim P(\theta)$.*

Then, for any training set $Z \sim \mathcal{D}^n$, R^* is a perfect, Bayes-optimal regularizer, where

$$R^*(\theta) = -\frac{1}{n} \log P(\eta(\theta)) .$$

The proof is given in the supplementary material.

In the special case where θ is the natural parameter (i.e., $\eta(\theta) = \theta$), Theorem 2 gives $R^*(\theta) = -\frac{1}{n} \log P(\theta)$. Using Bayes' rule, it can be shown that minimizing $\hat{L}(\theta) + R^*(\theta)$ is equivalent to maximizing the posterior probability of θ (i.e., performing MAP inference).

4.1. Example: Coin Flips

Suppose we have a collection of coins, where coin i comes up heads with unknown probability p_i. Given a training dataset consisting of outcomes from flipping each coin a certain number of times, we would like to produce a vector θ, where θ_i is an estimate of p_i, so as to minimize expected log loss on test data. This can be viewed as a highly simplified version of the problem of click-through rate prediction for online ads (e.g., see McMahan et al. (2013)).

For each coin, assume $p_i \sim \text{Beta}(\alpha, \beta)$ for some unknown constants α and β. Using the fact that the Bernoulli distribution is a member of the exponential family whose conjugate prior is the Beta distribution, and the fact that overall log loss is the sum of the log loss for each coin, we can prove the following as a corollary of Theorem 2.

Corollary 1. *A Bayes-optimal regularizer for the coin flip problem is given by*

$$\text{LogitBeta}(\theta) \equiv -\frac{1}{n} \sum_i \alpha \log(\theta_i) + \beta \log(1 - \theta_i) .$$

Observe that the LogitBeta regularizer is linear, with feature vector $\langle -\sum_i \log(\theta_i), -\sum_i \log(1 - \theta_i) \rangle$.

Given a large validation dataset with many independent coins, it can be shown that validation loss approaches expected test loss. Thus, in the limit, LearnLinReg is guaranteed to recover the optimal hyperparameters (the unknown α and β) when using this feature vector.

5. Experiments

We now evaluate the LearnLinReg and TuneReg algorithms experimentally, using both real and synthetic data. Code for both algorithms is available on GitHub (Streeter, 2019).

5.1. Optimization Problems

We consider three optimization problems. The first is an instance of the coin bias estimation problem discussed in §4.1, with $N = 10^5$ coins whose true bias is drawn from a uniform distribution (a Beta distribution with $\alpha = \beta = 1$). Our training data is the outcome of a *single* flip for each coin, and we compute the test loss $L(\theta)$ exactly.

We then consider two problems that make use of deep networks trained on MNIST and ImageNet (Russakovsky et al., 2015). For MNIST, we train a convolutional neural network using a variant of the LeNet architecture (LeCun et al., 1998). We then consider a softmax regression problem that involves retraining only the last layer of the network. This is a convex optimization problem that we can solve exactly, allowing us to focus on the impact of the regularizer without worrying about the confounding effects of early stopping.

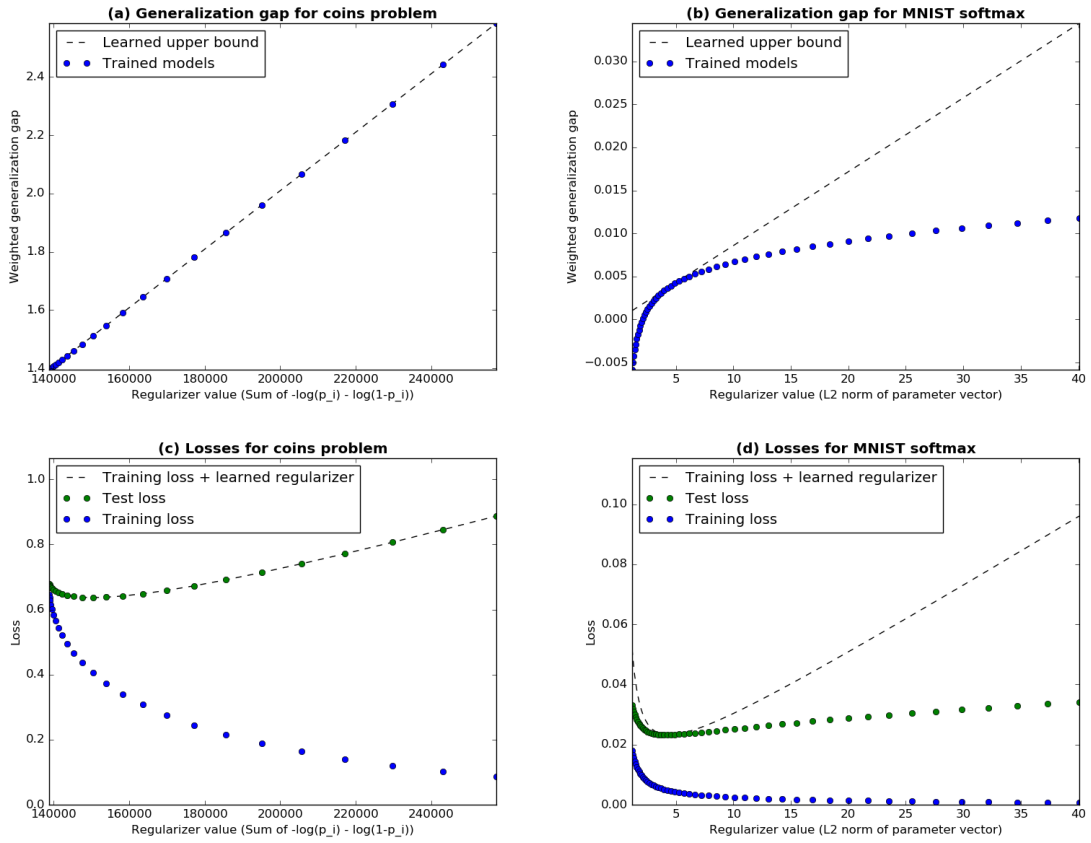

Figure 2. Estimates of the generalization gap for two optimization problems, and corresponding loss functions. For the coins problem, the LogitBeta regularizer perfectly estimates the generalization gap for all models. In contrast, for the MNIST softmax regression problem, L2 regularization provides an upper bound that is only tight for near-optimal models.

We then consider a transfer learning problem. Starting with an Inception-v3 model trained on ImageNet, we adapt the model to classify images of flowers from a public dataset (The TensorFlow Team, 2019) split evenly into training and test, retraining the last layer as in (Donahue et al., 2014).

5.2. Comparison of Regularizers

Which regularizers give the tightest generalization bounds? Does tightness in terms of slack translate into good test set performance?

To answer these questions, we solve the optimization problems discussed in §5.1 using several different regularizers. For the coins problem, we use the optimal LogitBeta regularizer from Corollary 1, with $\alpha = \beta = 1$. For the two softmax regression problems we use L1, L2, label smoothing, and a linearized version of dropout. For label smoothing, the loss function is equivalent to $\hat{L}(\theta) + \lambda R(\theta)$, where $R(\theta)$ is average training loss on a set of uniformly-labeled examples. For dropout, the regularizer is the difference between perturbed and unperturbed training loss, with the dropout probability fixed at .5.

For each problem, we proceed as follows. For each regularizer R_i, and for each λ in a predefined grid Λ_i, we generate a model $\hat{\theta}_{i,\lambda}$ by minimizing $\hat{L}(\theta) + \lambda R_i(\theta)$. For the coins problem, the solution can be found in closed form, while for the softmax regression problems, we obtain a near-optimal solution by running AdaGrad for 100 epochs. Each grid Λ_i contains 50 points. For L1 and L2, the grid is log-uniformly spaced over $[.1, 100]$, while for label smoothing and dropout it is uniform over $[0, 1]$. The result is a set of models, $\Theta_i \equiv \left\{\hat{\theta}_{i,\lambda} \mid \lambda \in \Lambda_i\right\}$.

For each regularizer R_i, and each $\theta \in \Theta_i$, we compute the training loss $\hat{L}(\theta)$, validation loss $V(\theta)$, and regularizer value $R_i(\theta)$. We then use LearnLinReg to compute an upper bound on validation loss, using $q_i(\theta) \equiv \langle R_i(\theta) \rangle$ as the feature vector. This produces a function of the form $f_i(\theta) = \hat{L}(\theta) + \lambda_i^* R_i(\theta)$ such that $f_i(\theta) \geq \alpha_i V(\theta)\ \forall \theta \in \Theta_i$.

Figure 2 shows the learned upper bounds for two combinations of problem and regularizer. In all graphs, the horizontal

Table 1. Comparison of regularizers, in terms of slack (see Figure 1), test loss, and test accuracy.

PROBLEM	REGULARIZER	MAX. SLACK	MAX. ADJ. SLACK	MIN. TEST LOSS	MAX. TEST ACCURACY
COINS($\lambda^* = 1$)	LOGITBETA	**0**	**0**	**0.637**	–
MNIST SOFTMAX REGRESSION	L1	5.83E-1	1.65E-3	0.0249	99.28%
	L2	1.70	**1.95e-4**	**0.0233**	99.31%
	LABEL SMOOTHING	**1.78e-1**	2.24E-3	0.0254	99.31%
	DROPOUT (LINEARIZED @ $p = .5$)	6.98E+01	2.92E-2	0.0463	**99.34%**
INCEPTION-V3 TRANSFER LEARNING	L1	**1.15**	2.36E-2	0.309	90.08%
	L2	1.47	**1.11e-4**	**0.285**	**90.74%**
	LABEL SMOOTHING	5.03	8.09E-2	0.366	89.65%
	DROPOUT (LINEARIZED @ $p = .5$)	2.52E+1	2.60E-1	0.506	90.46%

axis is the regularizer value $R_i(\theta)$. The top two graphs compare the weighted generalization gap, $\alpha_i V(\theta) - \hat{L}(\theta)$, to the learned upper bound $f_i(\theta) - \hat{L}(\theta) = \lambda_i^* R_i(\theta)$. For the coins problem (Figure 2 (a)), the learned upper bound is tight for all models, and perfectly predicts the generalization gap. In contrast, for the MNIST softmax regression problem with L2 regularization (Figure 2 (b)), the learned upper bound is only tight at near-optimal models.

Figure 2 (c) and (d) show the corresponding validation loss and (regularized) training loss. In both cases, the argmin of validation loss is very close to the argmin of regularized training loss when using the learned regularization strength, λ_i^*. We see qualitatively similar behavior for the other regularizers on both softmax regression problems.

Table 1 compares the upper bounds provided by each regularizers in terms of maximum slack and suboptimality-adjusted slack. To make the numbers comparable, the maximum is taken over *all* trained models (i.e., all $\theta \in \bigcup_i \Theta_i$). We also show the minimum test loss and maximum accuracy achieved using each regularizer. Observe that:

- Except for the coins problem, none of the regularizers produces a upper bound whose maximum slack is low (relative to test loss). However, L2 regularization achieves uniformly low *suboptimality-adjusted* slack.
- The rank ordering of the regularizers in terms of minimum test loss always matches the ordering in terms of maximum suboptimality-adjusted slack.
- Dropout achieves high accuracy despite poor slack and log loss, suggesting its role is somewhat different than that of the more traditional L1 and L2 regularizers.

5.3. Tuning Regularization Hyperparameters

The best values for hyperparameters are typically found empirically using black-box optimization. For regularization hyperparameters, TuneReg provides a potentially more efficient way to tune these hyperparameters, being guaranteed to "jump" to the optimal hyperparameter vector in just $k+1$ steps (where k is the number of hyperparameters) in the special case where a perfect regularizer exists. Does this theoretical guarantee translate into better performance on real-world hyperparameter tuning problems?

To answer this question, we compare TuneReg to random search and to Bayesian optimization using GP-EI-MCMC (Snoek et al., 2012), on each of the three optimization problems. For the coins problem, we use the known optimal LogitBeta regularizer, while for the two softmax regression problems we find a linear combination of the regularizers shown in Table 1 (L1, L2, label smoothing, and linearized dropout). For each problem, TuneReg samples the first $k+1$ points randomly, where k is the number of hyperparameters.

We consider two variants of each algorithm. In both cases, random search and TuneReg sample hyperparameter vectors uniformly from a hypercube, and GP-EI-MCMC uses this hypercube as its feasible set. In the first variant, the hypercube is based on the full range of hyperparameter values considered in the previous section. The second is a more "informed" variant based on data collected when generating Table 1: the feasible range for label smoothing (where applicable) is restricted to $[0, .1]$, and log scaling is applied to the L1, L2, and LogitBeta regularization hyperparameters. Using log scaling is equivalent to sampling from a log-uniform distribution for random search and TuneReg, and to tuning the log of the hyperparameter value for GP-EI-MCMC.

Figure 3 shows the best validation loss achieved by each algorithm as a function of the number of models trained. Each curve is an average of 100 runs. In all cases, TuneReg jumps to near-optimal hyperparameters on step $k+2$, whether or not the initial $k+1$ random points are sampled from the informative distribution ($k = 1$ in plot (a), and $k = 4$ in plots (b) and (c)). Both variants of TuneReg converge much faster than the competing algorithms, which typically require at least an *order of magnitude* more training runs in order to reach the same accuracy.

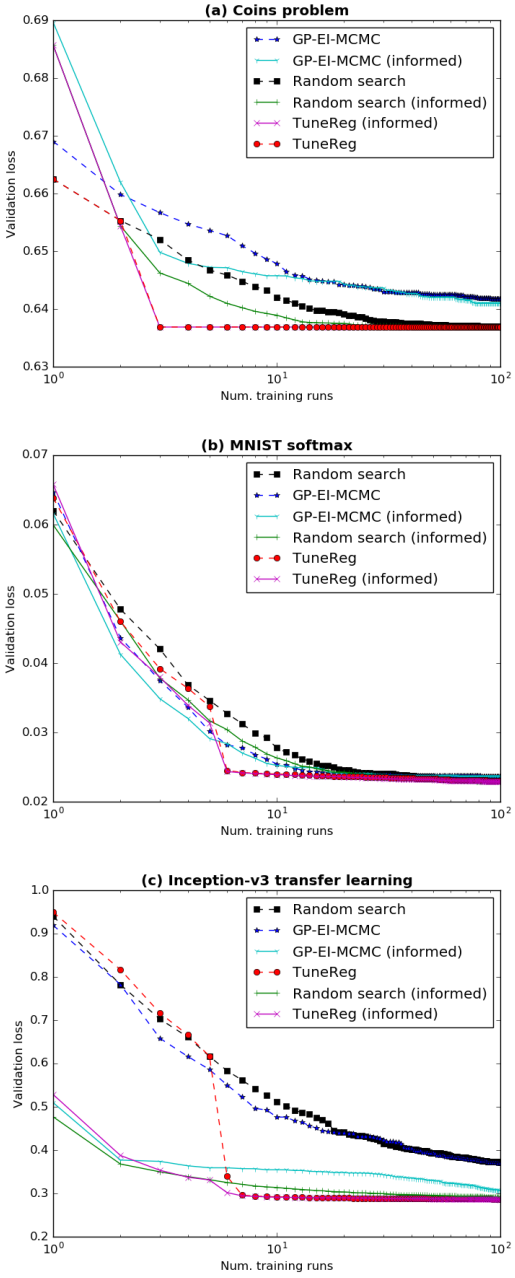

Figure 3. Comparison of algorithms for tuning regularization hyperparameters, with and without informative hyperparameter scaling and feasible range. For problems with k hyperparameters, TuneReg is able to "jump" to near-optimal hyperparameters after randomly sampling $k+1$ initial hyperparameter vectors ($k=1$ in plot (a), $k=4$ in plots (b) and (c)).

6. Related Work

The high-level idea that a good regularizer should provide an estimate of the generalization gap appears to be commonly known, though we are not aware of a specific source. What is novel in our work is the quantitative characterization of good regularizers in terms of slack and suboptimality, and the corresponding linear-programming-based algorithm for finding an approximately optimal regularizer.

Bounding the generalization gap is the subject of a vast literature, which has focused primarily on worst-case bounds (see Zhang et al. (2017) and references therein). These bounds have the advantage of holding with high probability for all models, but are typically too weak to be used effectively as regularizers. In contrast, our empirical upper bounds are only guaranteed to hold for the models we have seen, but are tight enough to yield improved generalization. Empirically, Jiang et al. (2019) showed that the generalization gap can be accurately predicted using a linear model based on margin information; whether this can be used for better regularization is an interesting open question.

An alternative to TuneReg is to use gradient-based methods which (approximately) differentiate validation loss with respect to the regularization hyperparameters (Pedregosa, 2016). Though this idea appears promising, current methods have not been shown to be effective on problems with more than one hyperparameter, and have not produced improvements as dramatic as the ones shown in Figure 3.

7. Conclusions

We have shown that the best regularizer is the one that gives the tightest bound on the generalization gap, where tightness is measured in terms of *suboptimality-adjusted slack*. We then presented the LearnLinReg algorithm, which computes approximately optimal hyperparameters for a linear regularizer using linear programming. Under certain Bayesian assumptions, we showed that LearnLinReg recovers an optimal regularizer given data from only $k+1$ training runs, where k is the number of hyperparameters. Building on this, we presented the TuneReg algorithm for tuning regularization hyperparameters, and showed that it outperforms state-of-the-art alternatives on both real and synthetic data.

Our experiments have only scratched the surface of what is possible using our high level approach. Promising areas of future work include (a) attempting to discover novel regularizers, for example by making a larger number of basis features available to the LearnLinReg algorithm, and (b) adjusting regularization hyperparameters on-the-fly during training, for example using an online variant of TuneReg.

References

Caruana, R., Lawrence, S., and Giles, C. L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13*, pp. 402–408, 2001.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. DeCAF: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 647–655, 2014.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. Predicting the generalization gap in deep networks with margin distributions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1222–1230. ACM, 2013.

Pedregosa, F. Hyperparameter optimization with approximate gradient. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 737–746. PMLR, 2016.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pp. 2951–2959, 2012.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Streeter, M. The LearnReg Library. https://github.com/google-research/google-research/tree/master/learnreg, 2019.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

The TensorFlow Team. Flowers, January 2019. URL http://download.tensorflow.org/example_images/flower_photos.tgz.

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