Boosted Backpropagation Learning for Training Deep Modular Networks

Alexander Grubb and J. Andrew Bagnell

June 2010
CMU-RI-TR-09-45
CMU-CS-09-172

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Divide-and-conquer is key to building sophisticated learning machines: hard problems are solved by composing a network of modules that solve simpler problems [13, 16, 4]. Many such existing systems rely on learning algorithms which are based on simple parametric gradient descent where the parametrization must be predetermined, or more specialized per-application algorithms which are usually ad-hoc and complicated. We present a novel approach for training generic modular networks that uses two existing. techniques: the error propagation strategy of backpropagation and more recent research on descent in spaces of functions [14, 18]. Combining these two methods of optimization gives a simple algorithm for training heterogeneous networks of functional modules using simple gradient propagation mechanics and established learning algorithms. The resulting separation of concerns between learning individual modules and error propagation mechanics eases implementation, enables a larger class of modular learning strategies, and allows **per-module** control of complexity/regularization. We derive and demonstrate this *functional backpropagation* and contrast it with traditional gradient descent in parameter space, observing that in our example domain the method is significantly more robust to local optima.

Keywords: boosting, Euclidean functional gradient, gradient descent, back-propagation, deep networks, ensemble methods

1 Introduction

For difficult learning problems that necessitate complex internal structure, it is common to use an estimator which is itself a network of simpler modules. Approaches leveraging such modular networks have been successfully applied to real-world problems like natural language processing (NLP) [11, 16], optical character recognition (OCR) [13, 8], and robotics [4, 20]. Figure 1 shows an example network for a robotic autonomy system where imitation learning is used for feedback.

These deep networks are typically composed of layers of learning modules with multiple inputs and outputs along with various transforming modules, e.g. the activation functions typically found in neural network literature, with the end goal of globally optimizing network behavior to perform a given task. These constructions offer a number of advantages over single learning modules, such as the ability to compactly represent highly non-linear hypotheses. A modular network is also a powerful method of representing and building in prior knowledge about problem structure and inherent sub-problems. [4]

Some approaches to network optimization rely on strictly local information, training each module using either synthetic or collected data specific to the function of that module. This is a common approach in NLP and vision systems, where modules correlate to individual tasks such as part of speech classification or image segmentation. The problem with this and other local training methods is the lack of end-to-end optimization of the system as a whole, which can lead to a compounding of errors and a degradation in performance. These local-only training algorithms can be useful as good initializations prior to global optimization, however.

A traditional approach to global network optimization is the well-studied technique of backpropagation [17, 19] which has been used for neural network training for over two decades. While initially used for training acyclic networks, extensions for recurrent and time-varying networks [19] have been developed. Backpropagation solves the problem of compounding errors between interacting modules by propagating error information throughout a network, allowing for end-to-end optimization with respect to a global measure of error. Further, it can be interpreted as a completely modular [3], object-oriented approach to semi-automatic differentiation that provides a separation of concerns between modules in the network.

Other approaches for complete optimization of networks [8, 10] have also shown promise as alternatives to backpropagation, but many of these algorithms are restricted to specific system architectures, and further, often rely upon a "fine-tuning" based on backpropagation. There have however been compelling results [2] as to the usefulness of using local module training as an initial optimization step, allowing for rapid learning prior to the traditionally slower global optimization step.

The basic backpropagation algorithm has previously been used to provide error signals for gradient descent in parameter space, sometimes making network optimization sensitive to the specific parametrization chosen. Recently,

powerful methods for performing gradient descent directly in a space of functions have been developed both for Reproducing Kernel Hilbert spaces [18] and for Euclidean function spaces [14, 6].

The former, known as kernel methods, are well studied in the literature and have been shown to be powerful means for learning non-linear hypotheses. The latter methods have been shown to be a generalization of the AdaBoost algorithm [5], another powerful non-linear learning method where complex hypotheses are built from arbitrary weak learners.

In the following sections, we present a method for combining functional gradient descent with backpropagation. Just as backpropagation allows a separation of concerns between modules, the proposed approach cleanly separates the problem of credit assignment for modules in the network from the problem of learning. This separation allows both a broader class of learning machines to be applied within the network architecture than standard backpropagation enables, and enables complexity control and generalization performance to be managed independently by each module in the network preventing the usual combinatorial search over all modules' internal complexities simultaneously. The approach further elucidates the notion of *structural local optima*—minima that hold in the space of functions and hence are tied to the modular structure—as contrasted with *parametric local optima* which are "accidents" of the chosen parameterization.

We have selected Euclidean functional gradients because of the flexibility provided in choosing base learners and the simplicity of implementing the algorithm in a modular manner. We begin by briefly reviewing Euclidean functional gradient descent, followed by the modified backpropagation algorithm for functional gradients. Following that we present a comparison of parameterized gradient descent and our functional gradient based method.

2 Euclidean Functional Gradient Descent

In the Euclidean function optimization setting, we seek to minimize a cost functional $\mathcal{R}[f]$, defined over a set of sampled points $\{\mathbf{x}_n\}_{n=1}^N$ and accompanying loss functions $\{l_n\}_{n=1}^N$ defined over possible predictions

$$\mathcal{R}[f] = \sum_{n=1}^{N} l_n(f(\mathbf{x}_n))$$

by searching over a Euclidean function space $\mathcal{F}$ (an L^2 space of square-integrable functions) of possible functions f.

The desired minimizing function for this cost functional can be found using a steepest descent optimization procedure in function space directly, in contrast to parameterized gradient descent where the gradient is evaluated with respect to the parameters of the function. The functional gradient of $\mathcal{R}[f]$ in this Euclidean

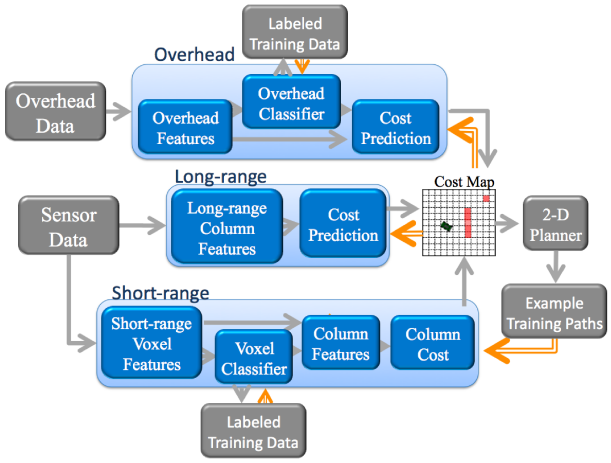

Figure 1: Modular network from the UPI perception and planning system for an off-road autonomous vehicle. Image courtesy [4].

function space is given as [7]:

$$\nabla_f \mathcal{R}[f] = \sum_{i=1}^{N} \nabla_f l_n(f(\mathbf{x}_n)) = \sum_{i=1}^{N} l_n'(f(\mathbf{x}_n))\delta_{\mathbf{x}_n}$$

using the chain rule and the fact that $\nabla_f f(\mathbf{x}_n) = \delta_{\mathbf{x}_n}$ [15], where $\delta_{\mathbf{x}_n}$ is the Dirac delta function centered at $\mathbf{x}_n$. The resulting gradient is itself a function composed of the sum of zero-width impulses centered at the points $\mathbf{x}_n$, scaled by the derivative $\frac{\partial^+ l}{\partial f(\mathbf{x}_n)}$.

Instead of using the explicit functional gradient as the direction for the gradient step, the gradient is projected onto a space of functions $\mathcal{H}$, to both allow for generalization and to constrain the search space to some reasonable hypothesis set. The resulting projected direction h^* can be found by minimizing the functional least squares projection of the gradient in L^2 function space [6, 15]:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \left[\sum_{n=1}^{N} (h(\mathbf{x}_n) - \nabla_f \mathcal{R}[f](\mathbf{x}_n))^2 \right] \tag{1}$$

which is equivalent to the familiar least squares regression problem over the dataset $\{\mathbf{x}_n, \nabla_f \mathcal{R}[f](\mathbf{x}_n)\}$.

These projected gradients are then used to repeatedly update the function, giving the gradient update rule for f as $f(\mathbf{x}) \leftarrow f(\mathbf{x}) - \alpha h^*(\mathbf{x})$. The final learned function is a sum of gradient steps over time

$$f(\mathbf{x}) = f_0(x) - \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

Algorithm 1 Projected Functional Gradient Descent

Given: initial function value f_0, step size schedule $\{\alpha_t\}_{t=1}^T$
for $t = 1, \ldots, T$ **do**
  Compute gradient $\nabla_f \mathcal{R}[f]$.
  Project gradient to hypothesis space $\mathcal{H}$ using least squares projection to find h^*.
  Update f: $f_t \leftarrow f_{t-1} - \alpha_t h^*$.
end for

where $h_t(x)$ is a function representing the gradient step taken at time t along with the corresponding step size α_t and starting point f_0. A brief description of this algorithm is given in Algorithm 1, in a manner that generalizes AdaBoost. [14, 6]

3 Backpropagation for Functional Gradients

Using the Lagrangian framework previously developed by LeCun [12], we now present the first part of our contribution: a derivation of backpropagation mechanics for functional gradients, both in Euclidean function space and reproducing kernel Hilbert space (RKHS). In this setting we have a layered network of functions f_k, $\mathbf{x}_{nk} = f_k(\mathbf{x}_{n(k-1)})$ where $n \in [1, N]$ indexes training examples and $k \in [1, K]$ indexes layers. Here $\mathbf{x}_{nk}$ represents the output of layer k for exemplar n, with $\mathbf{x}_{n0}$ defined to be the training input and $\mathbf{x}_{nK}$ the network output.

We seek to optimize a subset $F \subseteq \{f_k\}_{k=1}^K$ of these functions directly while the rest of the functions $f_k \notin F$ remain fixed. These fixed functions are arbitrary activation or intermediate transformation functions in the network, and can range from a simple sigmoid function to an A* planner.

The optimization of F is with respect to a set of loss functions defined over network outputs $l_n(\mathbf{x}_{nK})$. We can define the local Lagrange function for example n and the complete Lagrange function as

$$\begin{aligned} L_n(F, X_n, \Lambda_n) &= l_n(\mathbf{x}_{nK}) + \\ &\qquad \sum_{k=1}^{K} {\lambda_{nk}}^T (\mathbf{x}_{nk} - f_k(\mathbf{x}_{n(k-1)})) \\ L(F, X, \Lambda) &= \sum_{n=1}^{N} L_n(F, X_n, \lambda_n) \end{aligned}$$

with Lagrange multipliers λ_{nk} enforcing the forward propagation mechanics of the network.

As discussed by LeCun [12], $\nabla L(F, X, \Lambda) = 0$ is a necessary condition for any set of functions which are a stationary point with respect to the loss functions l_n while still satisfying the constraints. This results in three separate conditions

which must hold at the stationary point:

$$\frac{\partial L(F, X, \Lambda)}{\partial \Lambda} = \frac{\partial L(F, X, \Lambda)}{\partial X} = \frac{\partial L(F, X, \Lambda)}{\partial F} = 0 \tag{2}$$

3.1 Forward Propagation

Satisfying the first condition from (2) yields a separate constraint for each example n and layer k:

$$\begin{aligned}(2) &\implies \frac{\partial L(F, X, \Lambda)}{\partial \lambda_{nk}} = 0 \quad \forall n, k \\ &\implies \mathbf{x}_{nk} = f_k(\mathbf{x}_{n(k-1)}) \quad \forall n, k\end{aligned}$$

These constraints simply re-state the forward propagation mechanics of the network.

3.2 Backward Propagation

Similarly, satisfying the second part of (2) provides another set of constraints over the training data and layers:

$$\begin{aligned}(2) &\implies \frac{\partial L(F, X, \Lambda)}{\partial \mathbf{x}_{nk}} = 0 \quad \forall n, k \\ &\implies \lambda_{nK} = l'_n(\mathbf{x}_{nK}) \quad \forall n \\ &\qquad \lambda_{nk} = J_{f_{k+1}}(\mathbf{x}_{nk})\lambda_{n(k+1)} \quad \forall n, k < K\end{aligned}$$

where $J_f(X)$ is the Jacobian matrix of f at X.

These constraints define the mechanics for backwards error propagation. The Lagrange multipliers λ_{nk} store the accumulated results of applying the chain rule to the original derivatives of the loss function. Using the ordered derivative notation of Werbos [19], each element λ_{nki} represents the derivative of loss with respect to output x_{nki}, $\frac{\partial^+ l_n}{\partial x_{nki}}$.

3.3 Functional Gradient Update

The final condition in (2) gives a necessary constraint on the final optimized functions in F:

$$\begin{aligned}(2) &\implies \frac{\partial L(F, X, \Lambda)}{\partial f_k} = 0 \quad \forall f_k \in F \\ &\implies \nabla_f[L(F, X, \Lambda)] = 0 \quad \forall f_k \in F \\ &\implies \sum_{n=1}^{N} \lambda_{nk}(\nabla_f[f_k(\mathbf{x}_{n(k-1)})]) = 0 \quad \forall f_k \in F\end{aligned}$$

These constraints necessitate that each f_k must be a fixed point of the Lagrange equation L. Since we seek to minimize the loss functions, a steepest descent procedure can be used to find the minimum with function update rule:

$$f_k \leftarrow f_k - \alpha \sum_{n=1}^{N} \lambda_{nk} (\nabla_f [f_k(\mathbf{x}_{n(k-1)})]) \quad \forall f_k \in F$$

For RKHS function spaces the functional gradient of a function itself evaluated at $\mathbf{x}$ is the kernel centered at that point $K(\mathbf{x}, \cdot)$ [18]. Applying this to our functional update we get the following functional update rule:

$$f_k \leftarrow f_k - \alpha \sum_{n=1}^{N} \lambda_{nk} K(\mathbf{x}_{n(k-1)}, \cdot) \quad \forall f_k \in F$$

And for Euclidean function spaces (in the idealized case) the functional gradient of a function itself is again the Dirac delta function. Correspondingly, we get the following function update rule:

$$f_k \leftarrow f_k - \alpha \sum_{n=1}^{N} \lambda_{nk} \delta_{\mathbf{x}_{n(k-1)}} \quad \forall f_k \in F$$

In practice the equivalent projected version of this gradient step is used. This amounts to building a dataset $\{(\mathbf{x}_{n(k-1)}, \lambda_{nk})\}_{n=1}^{N}$ and using it to train a weak learner h^* as in (1).

3.4 Generalization to Other Network Topologies

The derivation here is presented for a sequential layered network, but it extends with no complications to directed acyclic graphs of modules. For any DAG, we can convert it into a layered network as above by first sorting the modules using a topological ordering and then modifying each layer to only accept values $\mathbf{z} \subseteq \mathbf{x}_{n(k-1)}$ that were originally used by that function: $\mathbf{x}_{nk} = (f_k(\mathbf{z}), \mathbf{x}_{n(k-1)}/\mathbf{z})$. The backwards propagation rules similarly only apply the Jacobian to a subset of the errors being passed back, while others are simply passed on in the topological ordering.

From there the derivation is fundamentally the same, with the functional update rule operating only on a subset of the inputs $\mathbf{x}_{n(k-1)}$ and error terms λ_{nk}. In a network of this form the backpropagation mechanics naturally follow the topology created in the network.

4 Implementation for a Modular Network

Using the formal derivation from the previous section, we now present an algorithm for training a series of boosted learning modules, by applying the standard boosting technique to functional backpropagation.

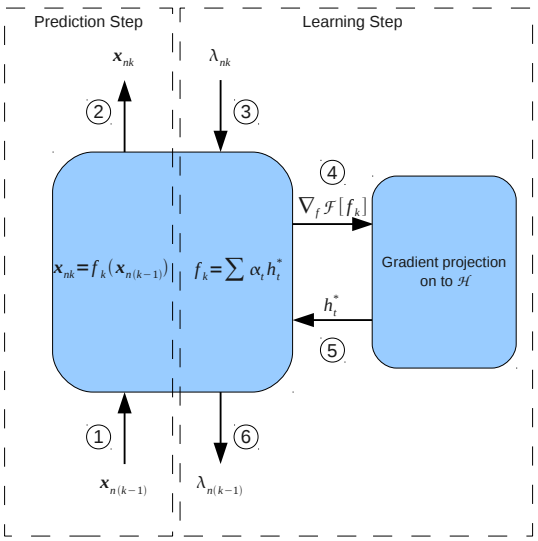

Figure 2: Example learning module illustrating backpropagation machinery and Euclidean functional gradient projection.

Algorithm 2 gives an outline for computing the forward and backward propagation steps for each functional learner in the network. The algorithm for training the complete network is the same as in backpropagation: a forward pass through the entire network is computed for the training data, the gradient of the loss function is evaluated, and then the backward pass propagates gradient information through the network and updates individual modules. Like any gradient-based procedure this can be repeated for a fixed number of steps or until some measure of convergence is reached.

Unlike standard boosting, there are some restrictions on the weak hypotheses which can be used. To accomodate the backpropagation of gradients, the functions in the hypothesis space $\mathcal{H}$ must be differentiable. Specifically we need to be able to calculate the Jacobian J_h for every function $h \in \mathcal{H}$. From there the Jacobian of each function f_k can be easily computed as they are all linear combinations of functions in $\mathcal{H}$.

This restriction does preclude some weak learners commonly employed in boosting, notably decision stumps, but still allows for a wide range of possible hypothesis spaces. If needed, this restriction can be relaxed for the first functional learner in a network as no gradient needs to be propagated through this layer.

A single functional module as described here is pictured in Figure 2. Each learning module is composed of machinery for computing the forward step and backward gradient propagation step, along with an internal gradient projection module which performs the boosting steps necessary to actually update the module.

Algorithm 2 Modular Functional Gradient Update

Functional Gradient Forward Step:
for all $\mathbf{x}_{n(k-1)}$ **do** {Step 1}
  Compute outputs $\mathbf{x}_{nk} = f_k(\mathbf{x}_{n(k-1)})$. {Step 2}
end for
Functional Gradient Backward Step:
for all λ_{nk} **do** {Step 3}
  Compute $\lambda_{n(k-1)} = J_{f_k}(\mathbf{x}_{n(k-1)})\lambda_{nk}$. {Step 6}
end for
Compute $\nabla_f L[f_k] = \lambda_{nk}\delta_{\mathbf{x}_{n(k-1)}}$. {Step 4} 1
Project gradient $\nabla_f L[f_k]$ on to $\mathcal{H}$ using least squares projection to find h_k^*. {Step 5}
Update f_k: $f_{kt} \leftarrow f_{k(t-1)} - \alpha_t h_k^*$.

4.1 Single Output Weak Learners

While the above formalism and algorithm consider each function f_k as a multi-output function, in practice it may be more convenient to treat each function f_k as being several single-output functions f_{kj} with outputs $\mathbf{x}_{nk} = (x_{nk1}, x_{nk2}, \ldots)$, where $x_{nkj} = f_{kj}(\mathbf{x}_{n(k-1)})$.

This is fundamentally equivalent to the multi-output formulation, but with the restriction that the hypothesis space $\mathcal{H}$ used for projection is itself a product of a given single-output hypothesis space $\mathcal{H} = \mathcal{G}^m$ where m is the output dimension. The gradient projection step in this restricted hypothesis space is equivalent to m independent projections over the datasets $\{(\mathbf{x}_{n(k-1)}, \lambda_{nkj})\}_{n=1}^N, \forall j$.

4.2 Online and Stochastic Learning

The literature on parametric gradient-based learning has shown that stochastic and online versions of the standard backpropagation algorithm are highly effective and convenient methods of learning, providing performance improvements and enabling practical learning from large or even infinite data sources. Both of these algorithms extend to the functional gradient versions of backpropagation presented here.

For Euclidean functional gradient boosting, while online learning on the per-example level is not feasible, an intuitive way of acheiving online behavior is to use "mini-batch" learning where a group of examples is collected or sampled from the underlying dataset and this small dataset is used for one iteration of the algorithm presented above. Using batches of examples is necessary in practice to obtain a reasonable and robust functional gradient projection.

In the RKHS setting, online learning easily generalizes and is a well studied problem in the literature [9].

4.3 Benefits of Modularity

This algorithm is inherently modular in two ways: it separates the individual pieces of the network from each other and it separates the structural aspects of the network from the learning in individual modules. This feature makes implementing complex networks of heterogenous modules straightforward and provides a number of mechanisms for improving learning performance.

In neural network-based architectures the complexity of the network is usually regulated by changing the structure of the network in some way. In contrast, the division between gradient propagation and gradient projection when using boosted backpropagation provides a means for varying the complexity of each layer without having to alter the structure of the network.

Another key benefit of the separate weak learners is that the local weak learners can use the gradient being projected to validate various local parameters, reducing the number of parameters and models that need to be globally optimized and validated. For example, if the weak learner being used is a regularized least squares regressor, the regularization parameter can be selected using the gradient dataset and cross-validation. This removes the need for an additional combinatorial search for regularization parameters at the global level, potentially saving a large amount of computation.

5 Experimental Results

5.1 Maximum Margin Planning (MMP)

Our first application is a simplified path planning system for an autonomous vehicle using Maximum Margin Planning (MMP) [15], a method for estimating optimal controllers which exhibit the same behavior as demonstrated human examples. The planning system, depicted in Figure 3, consists of feature extraction from overhead data, cost function mapping, and optimal planning (A*, here) modules. We seek to learn both the feature selection module, where raw terrain data is transformed into a set of high-level features, and a cost mapping function which takes the generated high-level features and produces costs appropriate for planning.

Formally, we are given a set of example maps $\mathcal{M}$ with locations in these maps $\mathbf{x}$ (essentially terrain feature examples). For the cost function module, we define the input $\phi(\mathbf{x})$ as the output of the feature extraction layer and then compute output $c(\phi(\mathbf{x}))$. The MMP cost functional $\mathcal{R}$ is defined as the difference between planned and demonstrated cost

$$\mathcal{R}[c] = \frac{1}{M}\sum_{i=1}^{M}\left(\sum_{\mathbf{x}\in\mathcal{M}_i} c(\phi(\mathbf{x}))\mu_i(\mathbf{x}) - \min_{\mu\in\mathcal{G}_i}\left\{\sum_{\mathbf{x}\in\mathcal{M}_i}(c(\phi(\mathbf{x})) - \ell_i(\mathbf{x}))\mu(\mathbf{x})\right\}\right)$$

where μ_i is the demonstrated path and the minimization $\min_{\mu \in \mathcal{G}_i}$ corresponds to the optimal path returned by the planning algorithm. This cost functional mathematically expresses the desired constraint that the behavior of the system after training duplicates the demonstrated behavior, by ensuring that the lowest cost paths in the examples are in fact the demonstrated paths. The extra $\ell_i(\mathbf{x})$ term corresponds to a margin function designed to ensure the demonstrated behavior is achieved by a significant margin. In our experiments we use $\ell_i(\mathbf{x}) = 0$ if $\mathbf{x}$ is on path μ_i and 1 otherwise.

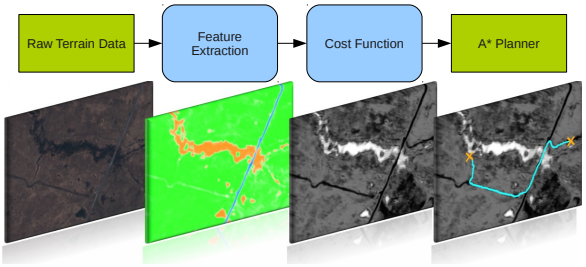

Figure 3: Maximum Margin Planning network for an autonomous vehicle. Learning modules are colored in blue, while the planning module is a fixed optimal planner. In this case, both a cost function and feature extraction layer are learned simultaneously to improve overall performance.

The cost functional as given does not appear to fit our previous model of a sum of individual loss functions l_n, but we can derive the appropriate initial backpropagation gradient by considering the functional gradient of $\mathcal{R}$ directly. This first functional gradient is equivalent to the first λ term from the formal derivation above.

Replacing the minimization with the actual optimal path according to the planner, μ_i^*, we get:

$$\nabla_f \mathcal{R}[c] = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{\mathbf{x} \in \mathcal{M}_i} \mu_i(\mathbf{x}) \delta_{\phi(\mathbf{x})} - \sum_{\mathbf{x} \in \mathcal{M}_i} \mu_i^*(\mathbf{x}) \delta_{\phi(\mathbf{x})} \right)$$

$$\nabla_f \mathcal{R}[c] = \sum_{\mathbf{x} \in \{\bigcup_{i=1}^{N} \mathcal{M}_i\}} \left(\frac{1}{N} (\mu_{i(\mathbf{x})}(\mathbf{x}) - \mu^*_{i(\mathbf{x})}(\mathbf{x})) \delta_{\phi(\mathbf{x})} \right)$$

Intuitively, this is equivalent to defining a loss function over outputs $y_{\mathbf{x}} = c(\phi(\mathbf{x}))$:

$$l_{\mathbf{x}}(y_{\mathbf{x}}) = y_{\mathbf{x}} \mu_i(\mathbf{x}) - (y_{\mathbf{x}} - \ell_i) \mu_i^*(\mathbf{x})$$
$$\text{where } i : \mathbf{x} \in \mathcal{M}_i$$

and using the same machinery formally outlined in Section 3.

Exponentiated Functional Gradient Descent. A number of empirical results in the MMP literature [15] have shown exponentiated functional gradient descent to be superior in performance, so we use this method of steepest

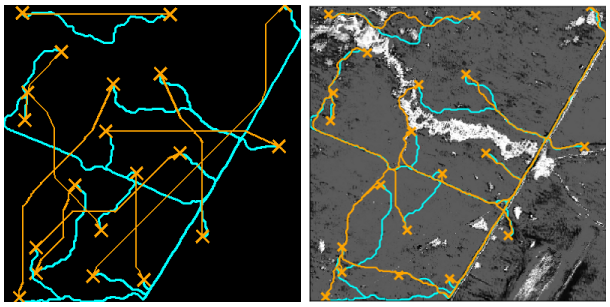

Figure 4: Performance on 10 test paths on an unseen map for a parameterized backpropagation (left) and a Euclidean functional gradient descent backpropagation (right) network. The parameterized version drives all costs to 0, resulting in a homogeneous map and straight-line paths.

descent for the costing module. The gradient is calculated in the same way as before, however instead of using an additive model as before, we now update the function $c(\cdot)$ using the appropriate exponentiated gradient rule:

$$c(\mathbf{x}) = e^{c_0(\mathbf{x})} \prod_{t=1}^{T} e^{\alpha_t h_t(\mathbf{x})}$$

Similar results can be derived for the parameterized gradient descent version of this network. In both cases the initial gradient passed in to the network is identical, and only the learning rule changes.

In the following experiments, the terrain features $\mathbf{x}$ are 5 by 5 patches of image data around each location taken from the satellite imagery. For the feature extraction module, $\phi(\mathbf{x})$, a two layer neural network was used in the parameterized gradient case, while an identically structured network using least squares linear regressors as weak learners was used in the functional gradient case.

5.1.1 Comparison of Parametric and Functional Gradients

Results for optimizing both networks using 4 example paths are found in Figures 4 and 5. In this instance, parameterized backpropagation gets caught in a severe local minimum early on while functional backpropagation achieves excellent performance.

We hypothesize that the poor performance of parameterized gradient descent is due to the larger number of negative gradient examples in the demonstrated path as compared to the planned path, driving costs down primarily. Essentially, the parametric version gets caught in this local minimum while trying to reduce the objective (the difference between example and planned path cost) by driving the costs of both paths down.

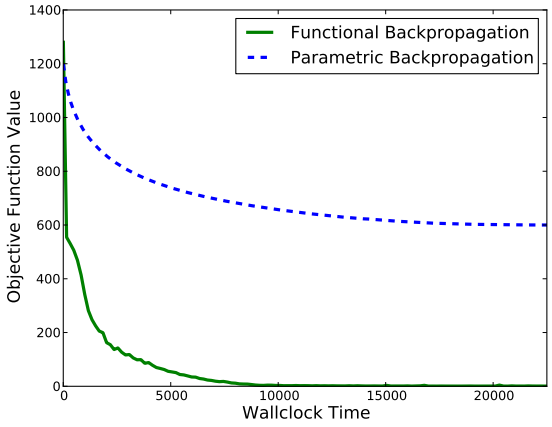

Figure 5: Plot of MMP objective function value for 4 training paths vs. wallclock time.

5.1.2 Local Parameter Validation

We also implemented a cross-validation based parameter selection method for determining the Tikhonov regularization parameter used in the linear least squares weak learners. Figure 6 shows the resulting parameter values as they were selected over time. Here we see small values initially, allowing for rapid initial learning, followed by relatively large values which prevent the network from overfitting. In contrast, we also performed a combinatorial search by hand for a good fixed regularization parameter. Figure 7 displays the final performance for both methods on both the example paths and paths on an unseen map.

Here the ability of the modular parameter validation to adjust over time is very beneficial, as we found that for small fixed global values initial learning is fast, but the algorithm is prone to overfitting, while with large fixed values the learning is slow to the point of making the optimization infeasible. The locally optimized parameters, however, allow for good initial behavior and generalization.

5.2 Classification

We also performed a set of comparison experiments on standard classification tasks. Our experiments were run on two widely used datasets, the MNIST handwritten digit dataset and the UCI letter recognition dataset [1].

For our experiments we used two-layer, fully connected networks of linear units, with a hyperbolic tangent activation function on the inner layers of units and a softmax activation function on the final layer of outputs. The cross entropy loss was the objective function to be minimized.

In the parameterized gradient case, this is simply a traditional two-layer neural network. For the boosted backpropagation network we use linear least

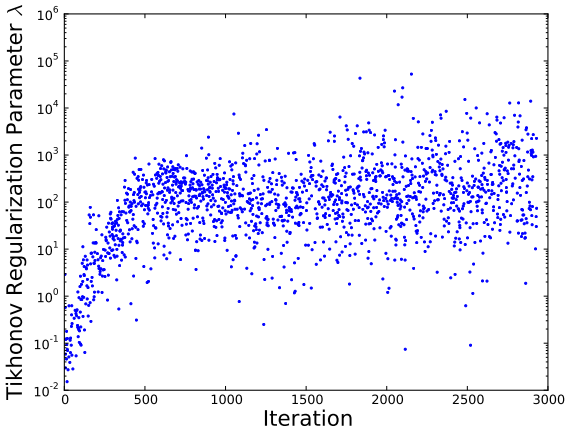

Figure 6: Locally optimized Tikhonov regularization parameter values for top layer of MMP network. Parameter selection was performed using cross-validation.

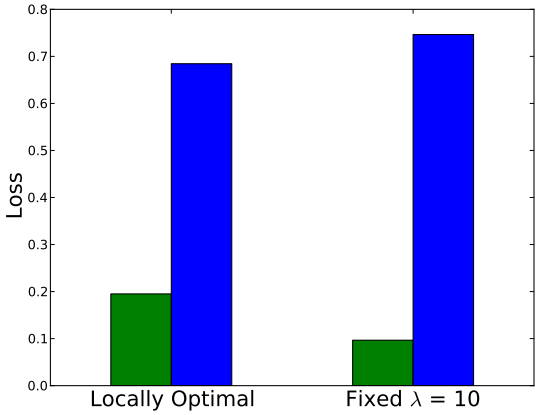

Figure 7: Comparison of training path (green) and test path (blue) performance for both locally optimized regularization and predetermined regularization parameters.

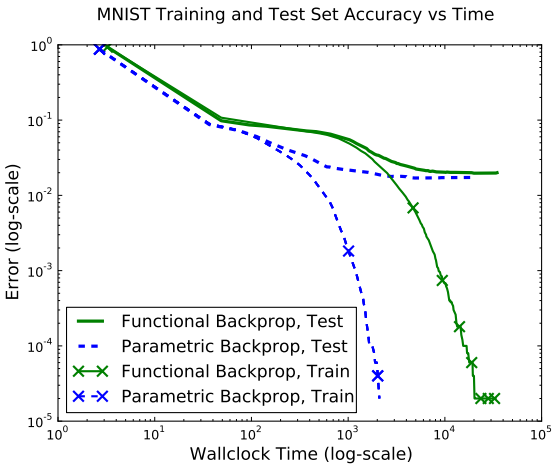

Figure 8: MNIST training and test set Error for both parameterized and functional backpropagation run on two-layer networks with 800 hidden units each.

Table 1: Test set error rates for MNIST and UCI Letter datasets.

Method	MNIST	UCI Letter
Parameterized Gradient	1.7%	8.6%
Functional Gradient	2.0%	7.5%

squares regression for gradient projection to facilitate comparison between the two methods.

Figures 8 and 9 show the error rates over time for both datasets, along with a summary of the test set performance in Table 1.

In the classification tasks, we find that the performance of the parametric and functional networks are roughly equivalent. The functional network performs slightly worse on the MNIST dataset, while it performs slightly better on the UCI Letter data set. On the MNIST dataset and its 784 input features, using unregularized linear least squares for the weak learner results in a severe overfitting of the data. It is here that the separation of learning logic from gradient propagation is most useful, as the regularization can be selected independently of the global network parameters, by using portions of the gradient being projected as a validation set; we found this both necessary and efficient for achieving good results. The UCI Letter dataset has far fewer input features at 16 and benefits less from the ability to separately regularize modules in the network. Here we see a small error decrease when using functional instead of parametric gradients.

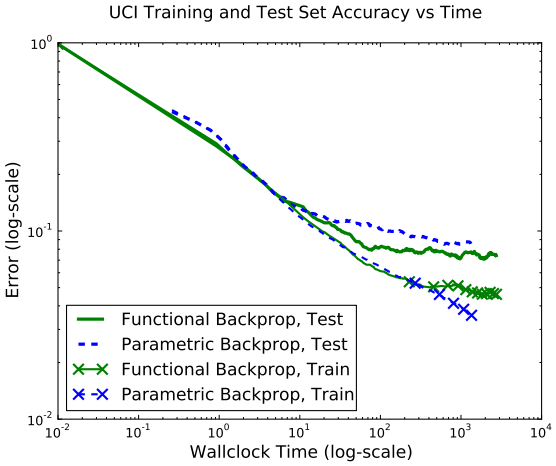

Figure 9: UCI Letter training and test set Error for both parameterized and functional backpropagation run on two-layer networks with 50 hidden units each.

6 Discussion and Future Work

We believe the combination of functional gradients with modular backpropagation provides significant promise. The separation of learning mechanism and structural error propagation in our method provides an important opportunity to keep learning local to an individual module, even in global network optimization. The ability to validate and perform model selection on each component network separately using error information may be crucial to efficiently implement the divide-and-conquer strategy modular systems are meant to use. Additionally, there is experimental indication that functional methods for network optimization provides a degree of robustness against parametric minima that occur when using complicated transformation modules like an optimal planner.

In this work, we largely focused on simple, linear weak learners to facilitate comparison with the parametric approach, although we have additional extensive experiments with non-linear learners. The non-linear methods offer the promise of greater system performance at a significantly larger computational expense. Future work will focus on achieving the benefits of these learning approaches while limiting the computational impact.

Acknowledgments

We would like to thank Daniel Munoz and Abraham Othman for their valuable feedback and David Bradley for his help with experimental work. This work is supported by the ONR MURI grant N00014-09-1-1052 and the Robotics Institute.

References

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[2] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[3] L. Bottou and P. Gallinari. A framework for the cooperation of learning algorithms. In *Advances in Neural Information Processing Systems*, pages 781–788, 1991.

[4] D. M. Bradley. *Learning in Modular Systems*. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2009.

[5] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. of the 13th International Conference on Machine Learning (ICML 1996)*, pages 148–156, 1996.

[6] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[7] I. M. Gelfand and S. V. Fomin. *Calculus of Variations*. Dover Publications, October 2000.

[8] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[9] J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Trans. on Signal Proc.*, 52(8):2165–2176, 2004.

[10] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10:1–40, 2009.

[11] S. Lawrence, C. L. Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Trans. on Knowl. and Data Eng.*, 12(1):126–140, 2000.

[12] Y. LeCun. A theoretical framework for back-propagation. In *Proc. of the 1988 Connectionist Models Summer School*, pages 21–28, 1988.

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

[14] L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*. MIT Press, 1999.

[15] N. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, July 2009.

[16] D. L. T. Rohde. *A connectionist model of sentence comprehension and production*. PhD thesis, Carnegie Mellon University, PA, USA, 2002.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Computational Models Of Cognition And Perception Series*, pages 318–362, 1986.

[18] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[19] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting.* Wiley-Interscience, New York, NY, USA, 1994.

[20] M. Zucker, J. A. Bagnell, C. Atkeson, and J. Kuffner. An optimization and learning approach to rough terrain locomotion. In *International Conference on Robotics and Automation, To Appear*, 2010.