GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks

Zhao Chen [1] **Vijay Badrinarayanan** [1] **Chen-Yu Lee** [1] **Andrew Rabinovich** [1]

Abstract

Deep multitask networks, in which one neural network produces multiple predictive outputs, can offer better speed and performance than their single-task counterparts but are challenging to train properly. We present a gradient normalization (GradNorm) algorithm that automatically balances training in deep multitask models by dynamically tuning gradient magnitudes. We show that for various network architectures, for both regression and classification tasks, and on both synthetic and real datasets, GradNorm improves accuracy and reduces overfitting across multiple tasks when compared to single-task networks, static baselines, and other adaptive multitask loss balancing techniques. GradNorm also matches or surpasses the performance of exhaustive grid search methods, despite only involving a single asymmetry hyperparameter α. Thus, what was once a tedious search process that incurred exponentially more compute for each task added can now be accomplished within a few training runs, irrespective of the number of tasks. Ultimately, we will demonstrate that gradient manipulation affords us great control over the training dynamics of multitask networks and may be one of the keys to unlocking the potential of multitask learning.

1. Introduction

Single-task learning in computer vision has enjoyed much success in deep learning, with many single-task models now performing at or beyond human accuracies for a wide array of tasks. However, an ultimate visual system for full scene understanding must be able to perform many diverse perceptual tasks simultaneously and efficiently, especially within the limited compute environments of embedded systems such as smartphones, wearable devices, and robots/drones. Such a system can be enabled by multitask learning, where one model shares weights across multiple tasks and makes multiple inferences in one forward pass. Such networks are not only scalable, but the shared features within these networks can induce more robust regularization and boost performance as a result. In the ideal limit, we can thus have the best of both worlds with multitask networks: more efficiency and higher performance.

In general, multitask networks are difficult to train; different tasks need to be properly balanced so network parameters converge to robust shared features that are useful across all tasks. Methods in multitask learning thus far have largely tried to find this balance by manipulating the forward pass of the network (e.g. through constructing explicit statistical relationships between features (Long & Wang, 2015) or optimizing multitask network architectures (Misra et al., 2016), etc.), but such methods ignore a key insight: *task imbalances impede proper training because they manifest as imbalances between backpropagated gradients*. A task that is too dominant during training, for example, will necessarily express that dominance by inducing gradients which have relatively large magnitudes. We aim to mitigate such issues at their root by directly modifying gradient magnitudes through tuning of the multitask loss function.

In practice, the multitask loss function is often assumed to be linear in the single task losses L_i, $L = \sum_i w_i L_i$, where the sum runs over all T tasks. In our case, we propose an adaptive method, and so w_i can vary at each training step t: $w_i = w_i(t)$. This linear form of the loss function is convenient for implementing gradient balancing, as w_i very directly and linearly couples to the backpropagated gradient magnitudes from each task. The challenge is then to find the best value for each w_i at each training step t that balances the contribution of each task for optimal model training. To optimize the weights $w_i(t)$ for gradient balancing, we propose a simple algorithm that penalizes the network when backpropagated gradients from any task are too large or too small. The correct balance is struck when tasks are training at similar *rates*; if task i is training relatively quickly, then its weight $w_i(t)$ should decrease relative to other task weights $w_j(t)|_{j \neq i}$ to allow other tasks more influence on

[1]Magic Leap, Inc. Correspondence to: Zhao Chen <zchen@magicleap.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

training. Our algorithm is similar to batch normalization (Ioffe & Szegedy, 2015) with two main differences: (1) we normalize across tasks instead of across data batches, and (2) we use rate balancing as a desired objective to inform our normalization. We will show that such gradient normalization (hereafter referred to as GradNorm) boosts network performance while significantly curtailing overfitting.

Our main contributions to multitask learning are as follows:

1. An efficient algorithm for multitask loss balancing which directly tunes gradient magnitudes.
2. A method which matches or surpasses the performance of very expensive exhaustive grid search procedures, but which only requires tuning a single hyperparameter.
3. A demonstration that direct gradient interaction provides a powerful way of controlling multitask learning.

2. Related Work

Multitask learning was introduced well before the advent of deep learning (Caruana, 1998; Bakker & Heskes, 2003), but the robust learned features within deep networks and their excellent single-task performance have spurned renewed interest. Although our primary application area is computer vision, multitask learning has applications in multiple other fields, from natural language processing (Collobert & Weston, 2008; Hashimoto et al., 2016; Søgaard & Goldberg, 2016) to speech synthesis (Seltzer & Droppo, 2013; Wu et al., 2015), from very domain-specific applications such as traffic prediction (Huang et al., 2014) to very general cross-domain work (Bilen & Vedaldi, 2017). Multitask learning has also been explored in the context of curriculum learning (Graves et al., 2017), where subsets of tasks are subsequently trained based on local rewards; we here explore the opposite approach, where tasks are jointly trained based on global rewards such as total loss decrease.

Multitask learning is very well-suited to the field of computer vision, where making multiple robust predictions is crucial for complete scene understanding. Deep networks have been used to solve various subsets of multiple vision tasks, from 3-task networks (Eigen & Fergus, 2015; Teichmann et al., 2016) to much larger subsets as in UberNet (Kokkinos, 2016). Often, single computer vision problems can even be framed as multitask problems, such as in Mask R-CNN for instance segmentation (He et al., 2017) or YOLO-9000 for object detection (Redmon & Farhadi, 2016). Particularly of note is the rich and significant body of work on finding explicit ways to exploit task relationships within a multitask model. Clustering methods have shown success beyond deep models (Jacob et al., 2009; Kang et al., 2011), while constructs such as deep relationship networks (Long & Wang, 2015) and cross-stich networks (Misra et al., 2016) give deep networks the capacity to search for meaningful relationships between tasks and to learn which features to share between them. Work in (Warde-Farley et al., 2014) and (Lu et al., 2016) use groupings amongst labels to search through possible architectures for learning. Perhaps the most relevant to the current work, (Kendall et al., 2017) uses a joint likelihood formulation to derive task weights based on the intrinsic uncertainty in each task.

3. The GradNorm Algorithm

3.1. Definitions and Preliminaries

For a multitask loss function $L(t) = \sum w_i(t) L_i(t)$, we aim to learn the functions $w_i(t)$ with the following goals: (1) to place gradient norms for different tasks on a common scale through which we can reason about their relative magnitudes, and (2) to dynamically adjust gradient norms so different tasks train at similar rates. To this end, we first define the relevant quantities, first with respect to the gradients we will be manipulating.

- W: The subset of the full network weights $W \subset \mathcal{W}$ where we actually apply GradNorm. W is generally chosen as the last shared layer of weights to save on compute costs[1].
- $G^{(i)}_W(t) = ||\nabla_W w_i(t) L_i(t)||_2$: the L_2 norm of the gradient of the weighted single-task loss $w_i(t)L_i(t)$ with respect to the chosen weights W.
- $\overline{G}_W(t) = E_{\text{task}}[G^{(i)}_W(t)]$: the average gradient norm across all tasks at training time t.

We also define various training rates for each task i:

- $\tilde{L}_i(t) = L_i(t)/L_i(0)$: the loss ratio for task i at time t. $\tilde{L}_i(t)$ is a measure of the *inverse* training rate of task i (i.e. lower values of $\tilde{L}_i(t)$ correspond to a faster training rate for task i)[2].
- $r_i(t) = \tilde{L}_i(t)/E_{\text{task}}[\tilde{L}_i(t)]$: the relative *inverse* training rate of task i.

With the above definitions in place, we now complete our description of the GradNorm algorithm.

3.2. Balancing Gradients with GradNorm

As stated in Section 3.1, GradNorm should establish a common scale for gradient magnitudes, and also should balance

[1] In our experiments this choice of W causes GradNorm to increase training time by only $\sim 5\%$ on NYUv2.

[2] Networks in this paper all had stable initializations and $L_i(0)$ could be used directly. When $L_i(0)$ is sharply dependent on initialization, we can use a theoretical initial loss instead. E.g. for L_i the CE loss across C classes, we can use $L_i(0) = \log(C)$.

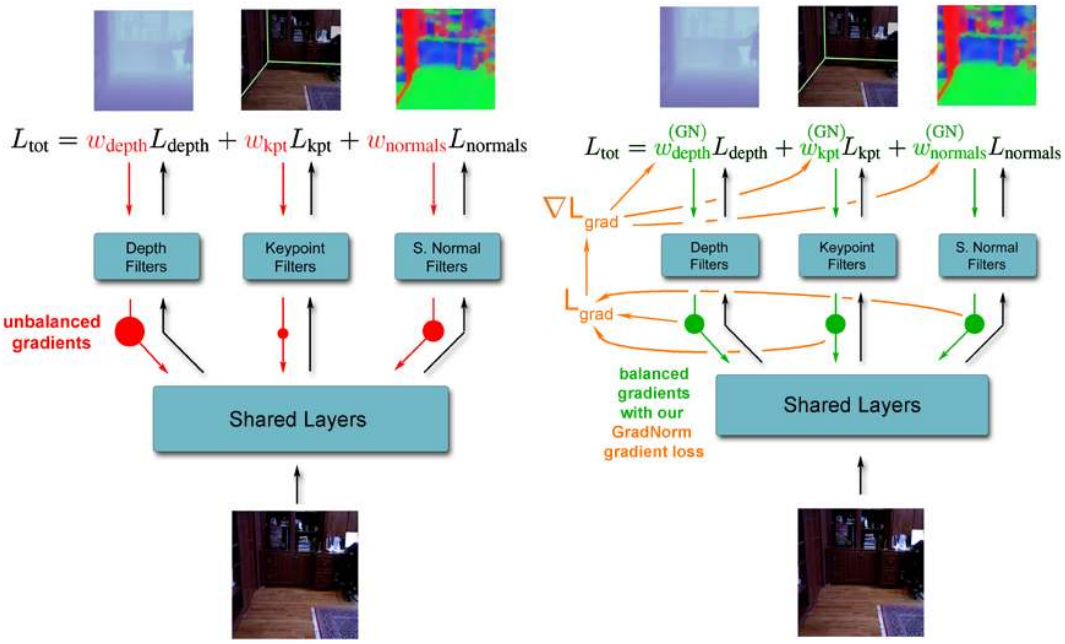

Figure 1. **Gradient Normalization.** Imbalanced gradient norms across tasks (left) result in suboptimal training within a multitask network. We implement GradNorm through computing a novel gradient loss L_{grad} (right) which tunes the loss weights w_i to fix such imbalances in gradient norms. We illustrate here a simplified case where such balancing results in equalized gradient norms, but in general there may be tasks that require relatively high or low gradient magnitudes for optimal training (discussed further in Section 3).

training rates of different tasks. The common scale for gradients is most naturally the average gradient norm, $\overline{G}_W(t)$, which establishes a baseline at each timestep t by which we can determine relative gradient sizes. The relative inverse training rate of task i, $r_i(t)$, can be used to rate balance our gradients. Concretely, the higher the value of $r_i(t)$, the higher the gradient magnitudes should be for task i in order to encourage the task to train more quickly. Therefore, our desired gradient norm for each task i is simply:

$$G^{(i)}_W(t) \mapsto \overline{G}_W(t) \times [r_i(t)]^\alpha, \tag{1}$$

where α is an additional hyperparameter. α sets the strength of the restoring force which pulls tasks back to a common training rate. In cases where tasks are very different in their complexity, leading to dramatically different learning dynamics between tasks, a higher value of α should be used to enforce stronger training rate balancing. When tasks are more symmetric (e.g. the synthetic examples in Section 4), a lower value of α is appropriate. Note that $\alpha = 0$ will always try to pin the norms of backpropagated gradients from each task to be equal at W. See Section 5.4 for more details on the effects of tuning α.

Equation 1 gives a target for each task i's gradient norms, and we update our loss weights $w_i(t)$ to move gradient norms towards this target for each task. GradNorm is then implemented as an L_1 loss function L_{grad} between the actual and target gradient norms at each timestep for each task, summed over all tasks:

$$L_{\text{grad}}(t; w_i(t)) = \sum_i \left| G^{(i)}_W(t) - \overline{G}_W(t) \times [r_i(t)]^\alpha \right|_1 \tag{2}$$

where the summation runs through all T tasks. When differentiating this loss L_{grad}, we treat the target gradient norm $\overline{G}_W(t) \times [r_i(t)]^\alpha$ as a fixed constant to prevent loss weights $w_i(t)$ from spuriously drifting towards zero. L_{grad} is then differentiated *only with respect to the* w_i, as the $w_i(t)$ directly control gradient magnitudes per task. The computed gradients $\nabla_{w_i} L_{\text{grad}}$ are then applied via standard update rules to update each w_i (as shown in Figure 1). The full GradNorm algorithm is summarized in Algorithm 1. Note that after every update step, we also renormalize the weights $w_i(t)$ so that $\sum_i w_i(t) = T$ in order to decouple gradient normalization from the global learning rate.

4. A Toy Example

To illustrate GradNorm on a simple, interpretable system, we construct a common scenario for multitask networks: training tasks which have similar loss functions but different loss scales. In such situations, if we naïvely pick $w_i(t) = 1$

Algorithm 1 Training with GradNorm

Initialize $w_i(0) = 1 \; \forall i$
Initialize network weights $\mathcal{W}$
Pick value for $\alpha > 0$ and pick the weights W (usually the final layer of weights which are shared between tasks)
for $t = 0$ **to** max_train_steps **do**
  Input batch x_i to compute $L_i(t) \; \forall i$ and $L(t) = \sum_i w_i(t) L_i(t)$ [standard forward pass]
  Compute $G^{(i)}_W(t)$ and $r_i(t) \; \forall i$
  Compute $\overline{G}_W(t)$ by averaging the $G^{(i)}_W(t)$
  Compute $L_{\text{grad}} = \sum_i |G^{(i)}_W(t) - \overline{G}_W(t) \times [r_i(t)]^\alpha|_1$
  Compute GradNorm gradients $\nabla_{w_i} L_{\text{grad}}$, keeping targets $\overline{G}_W(t) \times [r_i(t)]^\alpha$ constant
  Compute standard gradients $\nabla_{\mathcal{W}} L(t)$
  Update $w_i(t) \mapsto w_i(t+1)$ using $\nabla_{w_i} L_{\text{grad}}$
  Update $\mathcal{W}(t) \mapsto \mathcal{W}(t+1)$ using $\nabla_{\mathcal{W}} L(t)$ [standard backward pass]
  Renormalize $w_i(t+1)$ so that $\sum_i w_i(t+1) = T$
end for

for all loss weights $w_i(t)$, the network training will be dominated by tasks with larger loss scales that backpropagate larger gradients. We will demonstrate that GradNorm overcomes this issue.

Consider T regression tasks trained using standard squared loss onto the functions

$$f_i(\mathbf{x}) = \sigma_i \tanh((B + \epsilon_i)\mathbf{x}), \tag{3}$$

where $\tanh(\cdot)$ acts element-wise. Inputs are dimension 250 and outputs dimension 100, while B and ϵ_i are constant matrices with their elements generated IID from $\mathcal{N}(0, 10)$ and $\mathcal{N}(0, 3.5)$, respectively. Each task therefore shares information in B but also contains task-specific information ϵ_i. The σ_i are the key parameters within this problem; they are fixed scalars which set the scales of the outputs f_i. A higher scale for f_i induces a higher expected value of squared loss for that task. Such tasks are harder to learn due to the higher variances in their response values, but they also backpropagate larger gradients. This scenario generally leads to suboptimal training dynamics when the higher σ_i tasks dominate the training across all tasks.

To train our toy models, we use a 4-layer fully-connected ReLU-activated network with 100 neurons per layer as a common trunk. A final affine transformation layer gives T final predictions (corresponding to T different tasks). To ensure valid analysis, we only compare models initialized to the same random values and fed data generated from the same fixed random seed. The asymmetry α is set low to 0.12 for these experiments, as the output functions f_i are all of the same functional form and thus we expect the asymmetry between tasks to be minimal.

In these toy problems, we measure the *task-normalized* test-time loss to judge test-time performance, which is the sum of the test loss ratios for each task, $\sum_i L_i(t)/L_i(0)$. We do this because a simple sum of losses is an inadequate performance metric for multitask networks when different loss scales exist; higher loss scale tasks will factor disproportionately highly in the loss. There unfortunately exists no general single scalar which gives a meaningful measure of multitask performance in all scenarios, but our toy problem was specifically designed with tasks which are statistically identical except for their loss scales σ_i. There is therefore a clear measure of overall network performance, which is the sum of losses normalized by each task's variance σ_i^2 - equivalent (up to a scaling factor) to the sum of loss ratios.

For $T = 2$, we choose the values $(\sigma_0, \sigma_1) = (1.0, 100.0)$ and show the results of training in the top panels of Figure 2. If we train with equal weights $w_i = 1$, task 1 suppresses task 0 from learning due to task 1's higher loss scale. However, gradient normalization increases $w_0(t)$ to counteract the larger gradients coming from T_1, and the improved task balance results in better test-time performance.

The possible benefits of gradient normalization become even clearer when the number of tasks increases. For $T = 10$, we sample the σ_i from a wide normal distribution and plot the results in the bottom panels of Figure 2. GradNorm significantly improves test time performance over naïvely weighting each task the same. Similarly to the $T = 2$ case, for $T = 10$ the $w_i(t)$ grow larger for smaller σ_i tasks.

For both $T = 2$ and $T = 10$, GradNorm is more stable and outperforms the uncertainty weighting proposed by (Kendall et al., 2017). Uncertainty weighting, which enforces that $w_i(t) \sim 1/L_i(t)$, tends to grow the weights $w_i(t)$ too large and too quickly as the loss for each task drops. Although such networks train quickly at the onset, the training soon deteriorates. This issue is largely caused by the fact that uncertainty weighting allows $w_i(t)$ to change without constraint (compared to GradNorm which ensures $\sum w_i(t) = T$ always), which pushes the global learning rate up rapidly as the network trains.

The traces for each $w_i(t)$ during a single GradNorm run are observed to be stable and convergent. In Section 5.3 we will see how the time-averaged weights $E_t[w_i(t)]$ lie close to the optimal static weights, suggesting GradNorm can greatly simplify the tedious grid search procedure.

5. Application to a Large Real-World Dataset

We use two variants of NYUv2 (Nathan Silberman & Fergus, 2012) as our main datasets. Please refer to the Supplementary Materials for additional results on a 9-task facial landmark dataset found in (Zhang et al., 2014). The standard NYUv2 dataset carries depth, surface normals, and semantic

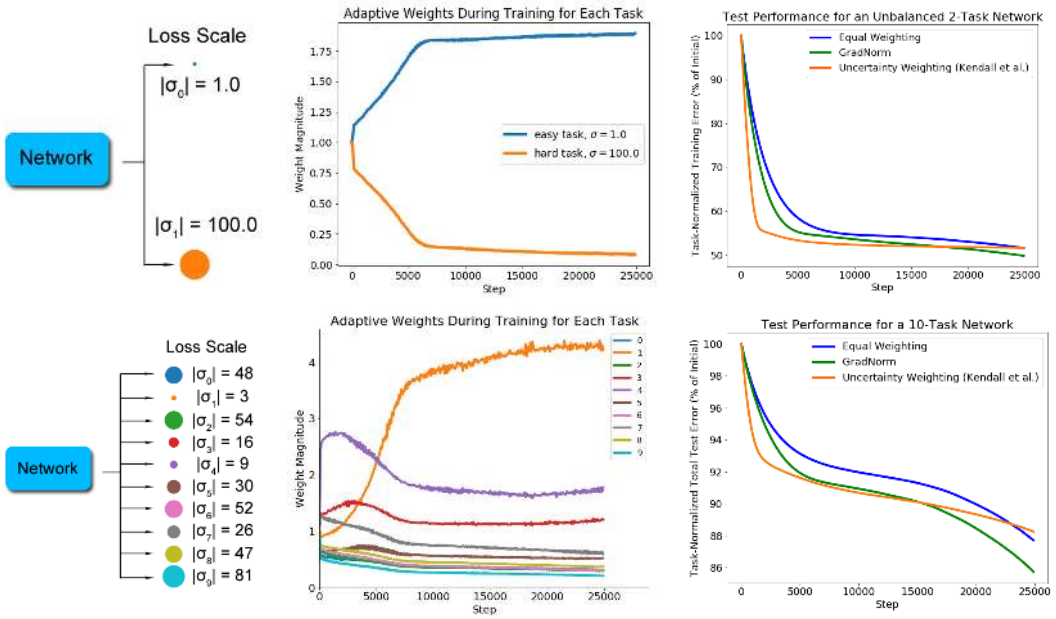

Figure 2. **Gradient Normalization on a toy 2-task (top) and 10-task (bottom) system.** Diagrams of the network structure with loss scales are on the left, traces of $w_i(t)$ during training in the middle, and task-normalized test loss curves on the right. $\alpha = 0.12$ for all runs.

segmentation labels (clustered into 13 distinct classes) for a variety of indoor scenes in different room types (bathrooms, living rooms, studies, etc.). NYUv2 is relatively small (795 training, 654 test images), but contains both regression and classification labels, making it a good choice to test the robustness of GradNorm across various tasks.

We augment the standard NYUv2 depth dataset with flips and additional frames from each video, resulting in 90,000 images complete with pixel-wise depth, surface normals, and room keypoint labels (segmentation labels are, unfortunately, not available for these additional frames). Keypoint labels are professionally annotated by humans, while surface normals are generated algorithmically. The full dataset is then split by scene for a 90/10 train/test split. See Figure 6 for examples. We will generally refer to these two datasets as NYUv2+seg and NYUv2+kpts, respectively.

All inputs are downsampled to 320 x 320 pixels and outputs to 80 x 80 pixels. We use these resolutions following (Lee et al., 2017), which represents the state-of-the-art in room keypoint prediction and from which we also derive our VGG-style model architecture. These resolutions also allow us to keep models relatively slim while not compromising semantic complexity in the ground truth output maps.

5.1. Model and General Training Characteristics

We try two different models: (1) a SegNet (Badrinarayanan et al., 2015; Lee et al., 2017) network with a symmetric VGG16 (Simonyan & Zisserman, 2014) encoder/decoder, and (2) an FCN (Long et al., 2015) network with a modified ResNet-50 (He et al., 2016) encoder and shallow ResNet decoder. The VGG SegNet reuses maxpool indices to perform upsampling, while the ResNet FCN learns all upsampling filters. The ResNet architecture is further thinned (both in its filters and activations) to contrast with the heavier, more complex VGG SegNet: stride-2 layers are moved earlier and all 2048-filter layers are replaced by 1024-filter layers. Ultimately, the VGG SegNet has 29M parameters versus 15M for the thin ResNet. All model parameters are shared amongst all tasks until the final layer. Although we will focus on the VGG SegNet in our more in-depth analysis, by designing and testing on two extremely different network topologies we will further demonstrate GradNorm's robustness to the choice of base architecture.

We use standard pixel-wise loss functions for each task: cross entropy for segmentation, squared loss for depth, and cosine similarity for normals. As in (Lee et al., 2017), for room layout we generate Gaussian heatmaps for each of 48 room keypoint types and predict these heatmaps with a pixel-wise squared loss. Note that all regression tasks are quadratic losses (our surface normal prediction uses a cosine loss which is quadratic to leading order), allowing us to use $r_i(t)$ for each task i as a direct proxy for each task's relative inverse training rate.

All runs are trained at a batch size of 24 across 4 Titan X GTX 12GB GPUs and run at 30fps on a single GPU at inference. All NYUv2 runs begin with a learning rate of 2e-5. NYUv2+kpts runs last 80000 steps with a learning rate

Table 1. Test error, NYUv2+seg for GradNorm and various baselines. Lower values are better. Best performance for each task is bolded, with second-best underlined.

Model and Weighting Method	Depth RMS Err. (m)	Seg. Err. (100-IoU)	Normals Err. (1-|cos|)
VGG Backbone			
Depth Only	1.038	-	-
Seg. Only	-	70.0	-
Normals Only	-	-	**0.169**
Equal Weights	0.944	70.1	0.192
GradNorm Static	<u>0.939</u>	**67.5**	<u>0.171</u>
GradNorm $\alpha = 1.5$	**0.925**	<u>67.8</u>	0.174

decay of 0.2 every 25000 steps. NYUv2+seg runs last 20000 steps with a learning rate decay of 0.2 every 6000 steps. Updating $w_i(t)$ is performed at a learning rate of 0.025 for both GradNorm and the uncertainty weighting ((Kendall et al., 2017)) baseline. All optimizers are Adam, although we find that GradNorm is insensitive to the optimizer chosen. We implement GradNorm using TensorFlow v1.2.1.

5.2. Main Results on NYUv2

In Table 1 we display the performance of GradNorm on the NYUv2+seg dataset. We see that GradNorm $\alpha = 1.5$ improves the performance of all three tasks with respect to the equal-weights baseline (where $w_i(t) = 1$ for all t,i), and either surpasses or matches (within statistical noise) the best performance of single networks for each task. The GradNorm Static network uses static weights derived from a GradNorm network by calculating the time-averaged weights $E_t[w_i(t)]$ for each task during a GradNorm training run, and retraining a network with weights fixed to those values. GradNorm thus can also be used to extract good values for static weights. We pursue this idea further in Section 5.3 and show that these weights lie very close to the optimal weights extracted from exhaustive grid search.

To show how GradNorm can perform in the presence of a larger dataset, we also perform extensive experiments on the NYUv2+kpts dataset, which is augmented to a factor of 50x more data. The results are shown in Table 2. As with the NYUv2+seg runs, GradNorm networks outperform other multitask methods, and either matches (within noise) or surpasses the performance of single-task networks.

Figure 3 shows test and training loss curves for GradNorm ($\alpha = 1.5$) and baselines on the larger NYUv2+kpts dataset for our VGG SegNet models. GradNorm improves test-time depth error by $\sim 5\%$, despite converging to a much higher training loss. GradNorm achieves this by aggressively rate balancing the network (enforced by a high asymmetry $\alpha = 1.5$), and ultimately suppresses the depth weight $w_{\text{depth}}(t)$ to lower than 0.10 (see Section 5.4 for more details). The same

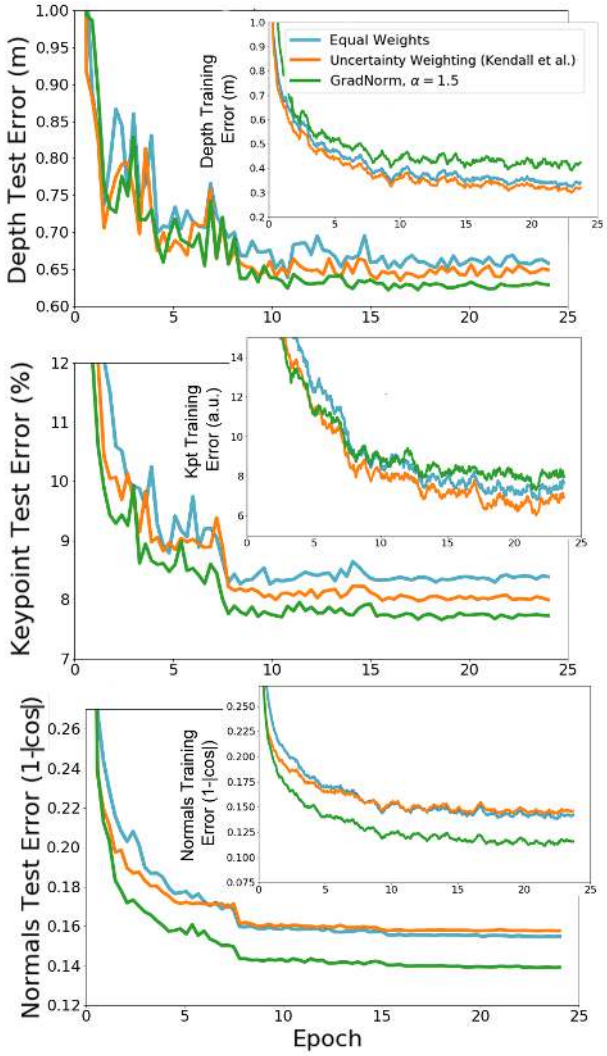

Figure 3. **Test and training loss curves for all tasks in NYUv2+kpts, VGG16 backbone**. GradNorm versus an equal weights baseline and uncertainty weighting (Kendall et al., 2017).

trend exists for keypoint regression, and is a clear signal of network regularization. In contrast, uncertainty weighting (Kendall et al., 2017) always moves test and training error in the same direction, and thus is not a good regularizer. Only results for the VGG SegNet are shown here, but the Thin ResNet FCN produces consistent results.

5.3. Gradient Normalization Finds Optimal Grid-Search Weights in One Pass

For our VGG SegNet, we train 100 networks from scratch with random task weights on NYUv2+kpts. Weights are sampled from a uniform distribution and renormalized to sum to $T = 3$. For computational efficiency, we only train for 15000 iterations out of the normal 80000, and then compare the performance of that network to our GradNorm

Table 2. Test error, NYUv2+kpts for GradNorm and various baselines. Lower values are better. Best performance for each task is bolded, with second-best underlined.

Model and Weighting Method	Depth RMS Err. (m)	Kpt. Err. (%)	Normals Err. (1-|cos|)
ResNet Backbone			
Depth Only	0.725	-	-
Kpt Only	-	7.90	-
Normals Only	-	-	**0.155**
Equal Weights	0.697	7.80	0.172
(Kendall et al., 2017)	0.702	7.96	0.182
GradNorm Static	0.695	7.63	0.156
GradNorm $\alpha = 1.5$	**0.663**	**7.32**	**0.155**
VGG Backbone			
Depth Only	0.689	-	-
Keypoint Only	-	8.39	-
Normals Only	-	-	0.142
Equal Weights	0.658	8.39	0.155
(Kendall et al., 2017)	0.649	8.00	0.158
GradNorm Static	0.638	**7.69**	**0.137**
GradNorm $\alpha = 1.5$	**0.629**	7.73	0.139

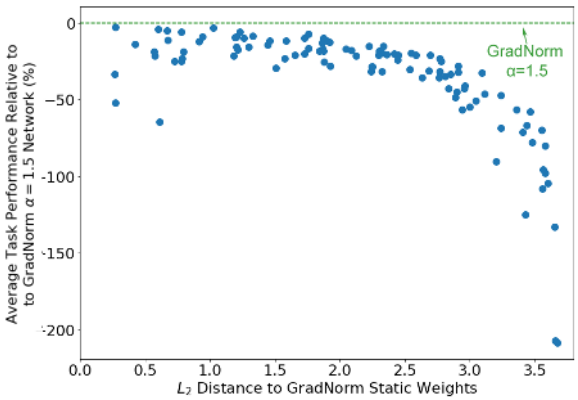

Figure 4. **Gridsearch performance for random task weights vs GradNorm, NYUv2+kpts**. Average change in performance across three tasks for a static multitask network with weights w_i^{static}, plotted against the L_2 distance between w_i^{static} and a set of static weights derived from a GradNorm network, $E_t[w_i(t)]$. A reference line at zero performance change is provided for convenience. All comparisons are made at 15000 steps of training.

$\alpha = 1.5$ VGG SegNet network at the same 15000 steps. The results are shown in Figure 4.

Even after 100 networks trained, grid search still falls short of our GradNorm network. Even more remarkably, there is a strong, negative correlation between network performance and task weight distance to our time-averaged GradNorm weights $E_t[w_i(t)]$. At an L_2 distance of ~ 3, grid search networks on average have almost double the errors per task compared to our GradNorm network. GradNorm has therefore *found the optimal grid search weights in one single training run*.

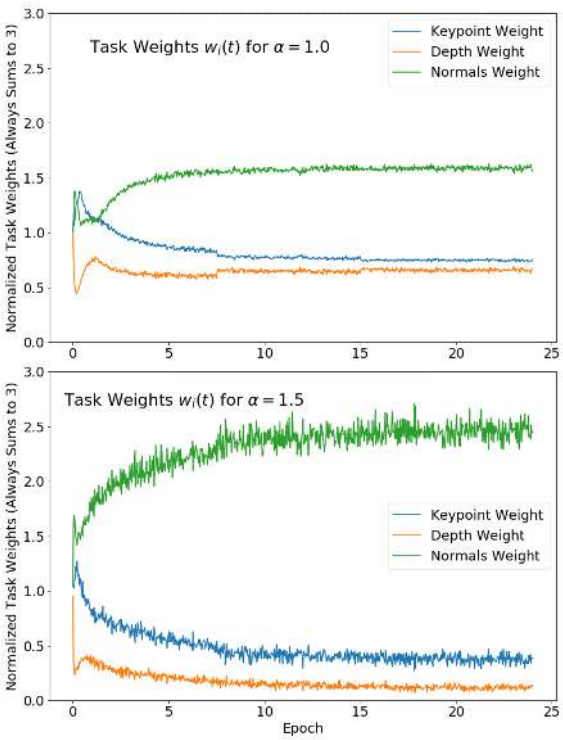

Figure 5. **Weights $w_i(t)$ during training, NYUv2+kpts.** Traces of how the task weights $w_i(t)$ change during training for two different values of α. A larger value of α pushes weights farther apart, leading to less symmetry between tasks.

5.4. Effects of tuning the asymmetry α

The only hyperparameter in our algorithm is the asymmetry α. The optimal value of α for NYUv2 lies near $\alpha = 1.5$, while in the highly symmetric toy example in Section 4 we used $\alpha = 0.12$. This observation reinforces our characterization of α as an asymmetry parameter.

Tuning α leads to performance gains, but we found that for NYUv2, almost any value of $0 < \alpha < 3$ will improve network performance over an equal weights baseline (see Supplementary for details). Figure 5 shows that higher values of α tend to push the weights $w_i(t)$ further apart, which more aggressively reduces the influence of tasks which overfit or learn too quickly (in our case, depth). Remarkably, at $\alpha = 1.75$ (not shown) $w_{\text{depth}}(t)$ is suppressed to below 0.02 at no detriment to network performance on the depth task.

5.5. Qualitative Results

Figure 6 shows visualizations of the VGG SegNet outputs on test set images along with the ground truth, for both the NYUv2+seg and NYUv2+kpts datasets. Ground truth labels are juxtaposed with outputs from the equal weights network, 3 single networks, and our best GradNorm network. Some

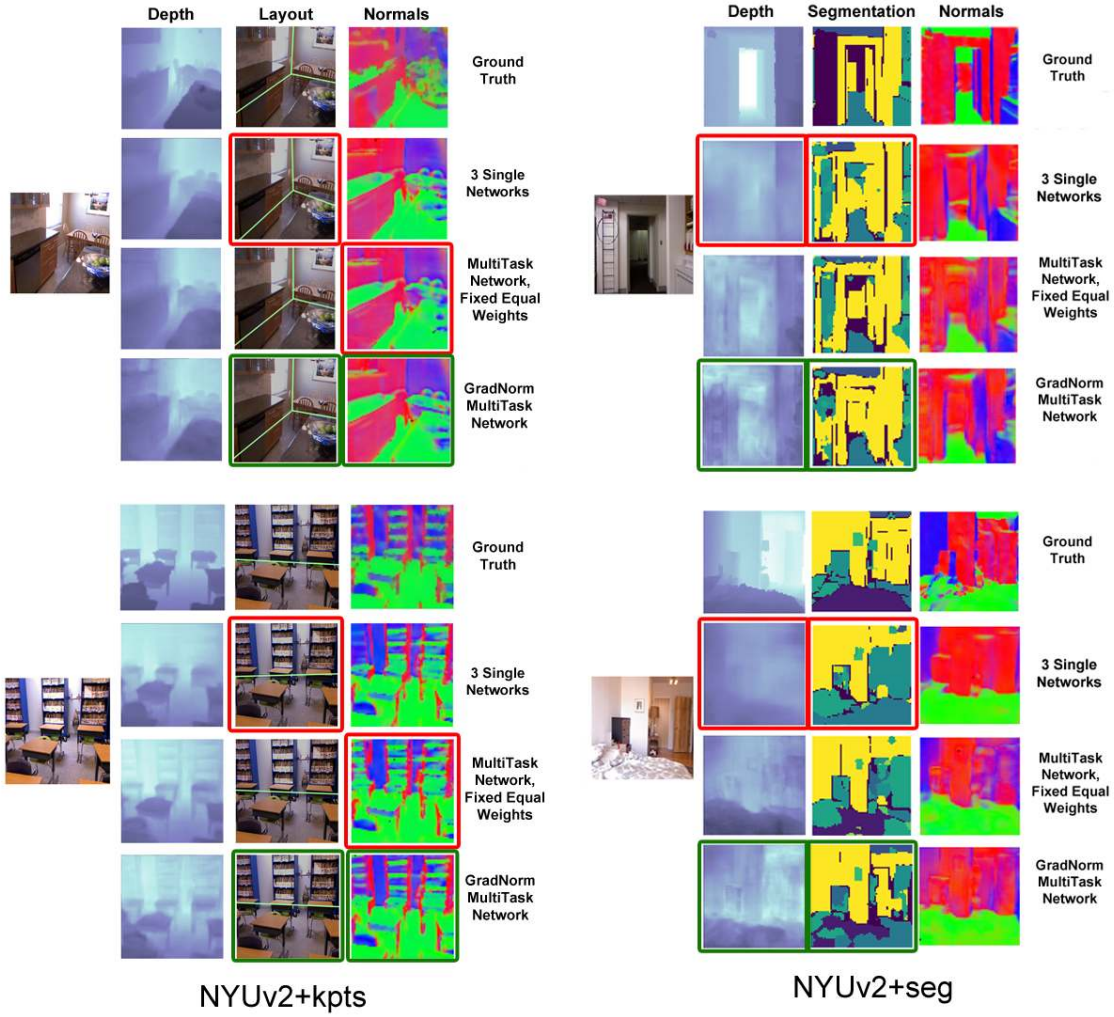

Figure 6. **Visualizations at inference time.** NYUv2+kpts outputs are shown on the left, while NYUv2+seg outputs are shown on the right. Visualizations shown were generated from random test set images. Some improvements are incremental, but red frames are drawn around predictions that are visually more clearly improved by GradNorm. For NYUv2+kpts outputs GradNorm shows improvement over the equal weights network in normals prediction and over single networks in keypoint prediction. For NYUv2+seg there is an improvement over single networks in depth and segmentation accuracy. These are consistent with the numbers reported in Tables 1 and 2.

improvements are incremental, but GradNorm produces superior visual results in tasks for which there are significant quantitative improvements in Tables 1 and 2.

6. Conclusions

We introduced GradNorm, an efficient algorithm for tuning loss weights in a multi-task learning setting based on balancing the training rates of different tasks. We demonstrated on both synthetic and real datasets that GradNorm improves multitask test-time performance in a variety of scenarios, and can accommodate various levels of asymmetry amongst the different tasks through the hyperparameter α. Our empirical results indicate that GradNorm offers superior performance over state-of-the-art multitask adaptive weighting methods and can match or surpass the performance of exhaustive grid search while being significantly less time-intensive.

Looking ahead, algorithms such as GradNorm may have applications beyond multitask learning. We hope to extend the GradNorm approach to work with class-balancing and sequence-to-sequence models, all situations where problems with conflicting gradient signals can degrade model performance. We thus believe that our work not only provides a robust new algorithm for multitask learning, but also reinforces the powerful idea that gradient tuning is fundamental for training large, effective models on complex tasks.

References

Badrinarayanan, V., Kendall, A., and Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

Bakker, B. and Heskes, T. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4(May):83–99, 2003.

Bilen, H. and Vedaldi, A. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv preprint arXiv:1701.07275*, 2017.

Caruana, R. Multitask learning. In *Learning to learn*, pp. 95–133. Springer, 1998.

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167. ACM, 2008.

Eigen, D. and Fergus, R. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2650–2658, 2015.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*, 2017.

Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

Huang, W., Song, G., Hong, H., and Xie, K. Deep architecture for traffic flow prediction: deep belief networks with multitask learning. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2191–2201, 2014.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Jacob, L., Vert, J.-p., and Bach, F. R. Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems*, pp. 745–752, 2009.

Kang, Z., Grauman, K., and Sha, F. Learning with whom to share in multi-task feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 521–528, 2011.

Kendall, A., Gal, Y., and Cipolla, R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv preprint arXiv:1705.07115*, 2017.

Kokkinos, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *arXiv preprint arXiv:1609.02132*, 2016.

Lee, C.-Y., Badrinarayanan, V., Malisiewicz, T., and Rabinovich, A. Roomnet: End-to-end room layout estimation. *arXiv preprint arXiv:1703.06241*, 2017.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

Long, M. and Wang, J. Learning multiple tasks with deep relationship networks. *arXiv preprint arXiv:1506.02117*, 2015.

Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., and Feris, R. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *arXiv preprint arXiv:1611.05377*, 2016.

Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3994–4003, 2016.

Nathan Silberman, Derek Hoiem, P. K. and Fergus, R. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Redmon, J. and Farhadi, A. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

Seltzer, M. L. and Droppo, J. Multi-task learning in deep neural networks for improved phoneme recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6965–6969. IEEE, 2013.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Søgaard, A. and Goldberg, Y. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 2, pp. 231–235, 2016.

Teichmann, M., Weber, M., Zoellner, M., Cipolla, R., and Urtasun, R. Multinet: Real-time joint semantic reasoning for autonomous driving. *arXiv preprint arXiv:1612.07695*, 2016.

Warde-Farley, D., Rabinovich, A., and Anguelov, D. Self-informed neural network structure learning. *arXiv preprint arXiv:1412.6563*, 2014.

Wu, Z., Valentini-Botinhao, C., Watts, O., and King, S. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 4460–4464. IEEE, 2015.

Zhang, Z., Luo, P., Loy, C. C., and Tang, X. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision*, pp. 94–108. Springer, 2014.