# GRADNORM: GRADIENT NORMALIZATION FOR ADAPTIVE LOSS BALANCING IN DEEP MULTITASK NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Deep multitask networks, in which one neural network produces multiple predictive outputs, are more scalable and often better regularized than their single-task counterparts. Such advantages can potentially lead to gains in both speed and performance, but multitask networks are also difficult to train without finding the right balance between tasks. We present a novel gradient normalization (GradNorm) technique which automatically balances the multitask loss function by directly tuning the gradients to equalize task training rates. 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 over single 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 $\alpha$. Thus, what was once a tedious search process which 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 hope to demonstrate that direct 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 models now performing at or beyond human accuracies for a wide array of tasks. However, a system that strives for full scene understanding cannot focus on one problem, but needs to perform many diverse perceptual tasks simultaneously. Such systems must also be efficient, especially within the restrictions of limited compute environments in embedded systems such as smartphones, wearable devices, and robots/drones. Multitask learning most naturally lends itself to this problem by sharing weights amongst different tasks within the same model and producing multiple predictions in one forward pass. Such networks are not only scalable, but the shared features within these networks tend to be better regularized and boost performance as a result. In the ideal limit, we can thus have the best of both worlds with multitask networks: both more efficiency and higher performance.

The key difficulty in multitask learning lies in the balancing of tasks, and perhaps the simplest way to control this balance is to choose the correct joint loss function. In practice, the multitask loss function is often assumed to be linear in the single task losses, $L = \sum_i w_i L_i$, where the sum runs over $T$ tasks. The challenge is then to find the best value for each $w_i$ that balances the contribution of each task for optimal model training. Our proposed method is furthermore an adaptive method, allowing $w_i$ to vary with the training time $t$, and so $w_i = w_i(t)$.

Our key insight lies in the observation that these $w_i(t)$ influence training only because they control the magnitude of the gradients generated from task $i$. As such, manipulating the gradient norms themselves would be a more direct way to control the training dynamics. More specifically, we propose a simple heuristic 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 to allow other tasks

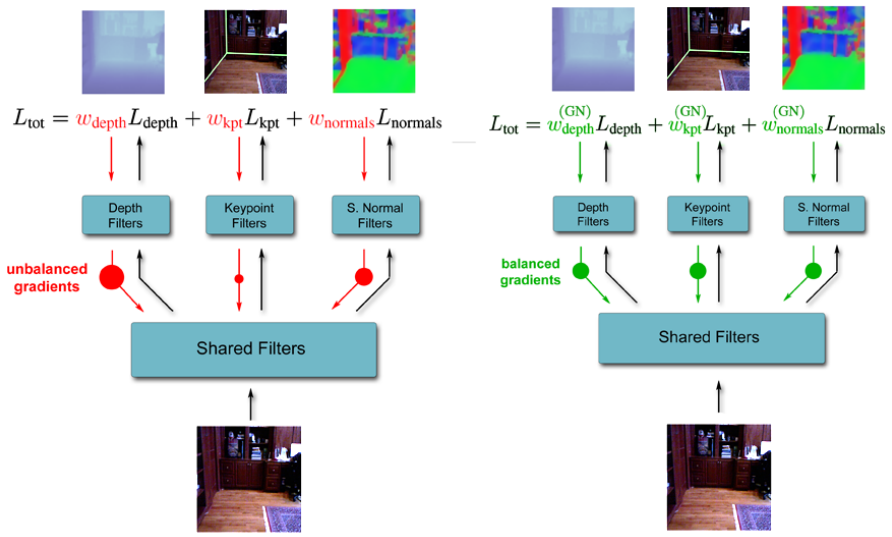


Figure 1: **Gradient Normalization.** We illustrate here the simple case when all gradient norms are equalized. In practice, tasks which tend to train more quickly will have their gradients suppressed.

more influence on the network. Our method can be said to be a form of batch normalization (Ioffe & Szegedy (2015)) for backpropagation, ensuring that gradients from each task per batch lie on a common statistical scale. We will show that, when implemented, gradient normalization leads to across-the-board improvements in accuracy and suppresses overfitting.

Our main contributions to the field of multitask learning are as follows:

1. An attractively simple heuristic for multitask loss balancing involving training rate equalization, which is implemented through a novel gradient loss function.
2. A simplification to exhaustive grid search (which has compute complexity $\mathcal{O}(N^T)$ for $N$ grid points in one dimension) that only involves tuning one robust hyperparameter.
3. Demonstration that direct interaction with gradients provides a powerful way of reasoning about multitask learning.

## 2 Related Work

Multitask learning has existed well before the advent of deep learning (Caruana (1998); Bakker & Heskes (2003)), but the robust learned features within deep networks have spurned renewed interest. Although our primary application area is computer vision, multitask learning has applications in multiple other fields, from natural language processing (Hashimoto et al. (2016); Collobert & Weston (2008); Søgaard & Goldberg (2016)) to speech synthesis (Wu et al. (2015); Seltzer & Droppo (2013)), from very domain-specific applications like traffic prediction (Huang et al. (2014)) to very general cross-domain work (Bilen & Vedaldi (2017)).

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)). Researchers often assume a fixed loss function or network architecture, but there has also been significant work on finding optimal ways to relate tasks to each other in a multitask model. Clustering methods have shown success beyond deep models (Kang et al. (2011); Jacob et al. (2009)), while constructs such as deep relationship networks (Long & Wang (2015)) and cross-stich networks (Misra et al. (2016)) search for meaningful relationships between tasks and learn which features to share between them.

Work by Lu et al. (2016) uses task groupings 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 METHODOLOGY

## 3.1 A GRADIENT LOSS FUNCTION BASED ON RATE BALANCING

Our goal is to *rate balance* our multitask network, ensuring that tasks train at similar rates. In our case, we measure training rate by the *loss ratio* of each task, $L'_i := L_i/L_i^{(0)}$, where $L_i^{(0)}$ is the initial loss of task $i$. This effectively initializes the gradients per task to all be the same magnitude at the start of training. We shall see this loss ratio helps balance both classification and regression tasks and can be easily adapted to other loss functions in Section 5.2.

Rate balancing is enforced at filter $F$ when:

$$|\nabla_F L_i| \propto \left( \frac{L_i/L_i^{(0)}}{\sum_j L_j/L_j^{(0)}} \right)^{\alpha} = \left( \frac{L'_i}{\sum_j L'_j} \right)^{\alpha}, \tag{1}$$

where the $\alpha$ hyperparameter can be thought of as a measure of the *asymmetry* between tasks. In cases where tasks are very different in their complexity, leading to different learning dynamics, a higher value of $\alpha$ should be used to pull the tasks back towards a common training rate more forcefully. When tasks are fully symmetric (e.g. the synthetic examples in Section 4), a lower value of $\alpha$ is appropriate. Note that $\alpha = 0$ will always try to pin the norms of backpropped gradients from each task to be equal at filter $F$.

Our desired proportionality in Equation 1 leads us naturally to the loss function for task $i$ at filter $F$:

$$L_{\text{grad}}^{(i)} = |\text{norm}(\nabla_F L_i) - \left( \frac{L'_i}{\sum_j L'_j} \right)^{\alpha} E_j(\text{norm}(\nabla_F L_j))|. \tag{2}$$

The expected value $E_j(\text{norm}(\nabla_F L_j))$ is a mean of the gradient norms of $L_j$ at filter $F$ across each task $j$. The above loss is for one task; the full loss is just the mean of the individual task losses, $L_{\text{grad}} = (1/T)\sum_i L_{\text{grad}}^{(i)}$. In simple terms, $\text{norm}(\nabla_F L_i)$ is pulled to the value of (relative training rate of task $i)^{\alpha} \times$(average gradient among all tasks). If the relative training rate is the same amongst all tasks (or if $\alpha = 0$), this loss pulls all gradient norms towards the mean gradient norm (Figure 1).

In practice, we only apply GradNorm to one downstream filter in our network, and $L_{\text{grad}}$ only updates the task weights $w_i(t)$ to simplify the problem; in our tests GradNorm only incurs a $\sim 5\%$ slowdown at training. Downstream normalization also should backpropagate benefits upstream. For convolutional filters, we separately consider the gradient norms of each filter channel, while for dense layers we consider the matrix norm of the full affine transformation matrix. Bias terms are left unnormalized. After every update, we 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 SIMPLE TOY EXAMPLE

To illustrate GradNorm on a simple system, we consider $T$ regression tasks onto the functions

$$f_i(\mathbf{x}) = \sigma_i \tanh(A_i \mathbf{x}), \tag{3}$$

where tanh acts element-wise and the matrices $A_i$ are generated from a common baseline matrix, $A_i = B + \epsilon_i$. The common baseline is crucial as it allows our model to find useful shared features. The $\sigma_i$ are fixed scalars which set the variance of the outputs $f_i$. Higher values of $\sigma_i$ induce higher values of squared loss for that task. These tasks are harder to learn due to the higher variances in their response values, but they also backpropagate larger gradients. Classically, such a scenario can lead to suboptimal training dynamics as the higher $\sigma_i$ tasks tend to dominate the training.

All toy problem runs use a 4-layer fully-connected ReLU-activated network with 100 neurons per layer as a common trunk. A final affine transformation per task gives $T$ final predictions. Inputs are

in $\mathbb{R}^{250}$, and the $A_i$ map inputs to $\mathbb{R}^{100}$. To ensure consistency, we only compare models initialized to the same random values and fed data generated from a fixed random seed. The asymmetry $\alpha$ is set low to 0.12 for these experiments, as the output functions $f_i$ are all of the same form.

In these toy problems, we measure the *task-normalized* test-time loss, which is the sum of the test loss ratios for each task, $\sum_i L'_i$. This metric is appropriate as it gives equal weight to all tasks.

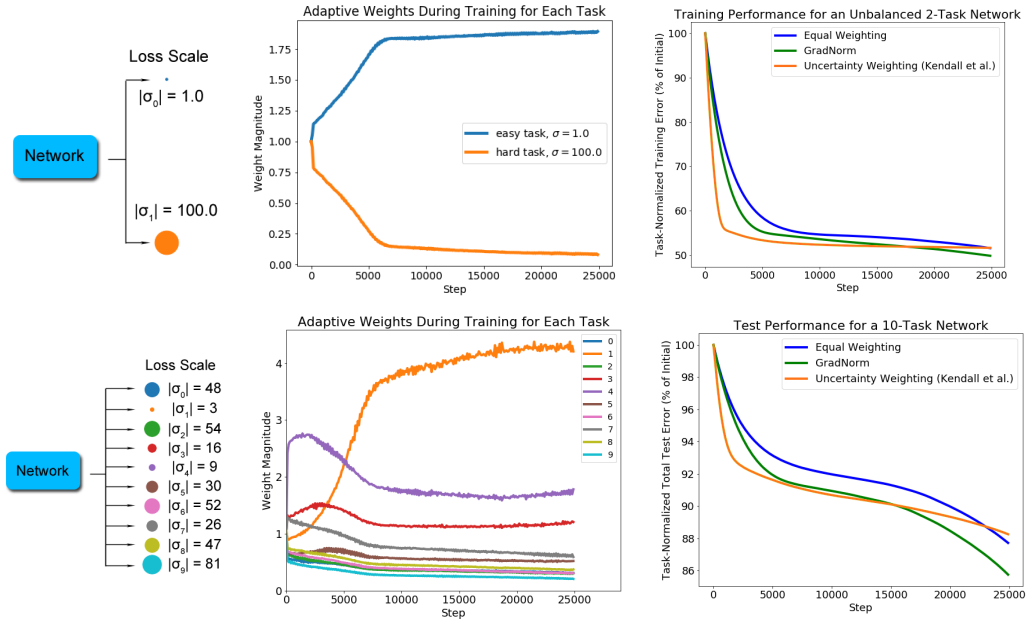


Figure 2: **Gradient Normalization on a toy 2-task (top) and 10-task (bottom) system.** Diagrams of the network structure with loss scales are on the left, traces of $w_i(t)$ during training in the middle, and task-normalized test loss curves on the right. $\alpha = 0.12$ for all runs.

In the case of $T = 2$, we choose the values $(\sigma_0, \sigma_1) = (1.0, 100.0)$. Classically, task 1 can suppress task 0's influence during training due to its higher loss scale. As shown in the top panels of Figure 2, gradient normalization remedies the issue by increasing $w_0(t)$ 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 $\sigma_i$ from a normal distribution and plot the results in the bottom row of Figure 2. GradNorm significantly improves test time performance over naively weighting each task the same. Like $T = 2$, for $T = 10$ the $w_i(t)$ grow larger for smaller $\sigma_i$ tasks; GradNorm is giving tasks with smaller loss scales more breathing room.

For both $T = 2$ and $T = 10$, GradNorm is more stable and outperforms the uncertainty weighting proposed by Kendall et al. (2017). Uncertainty weighting tends to grow weights too large and too quickly, so although networks train quickly at the onset the training is less stable in the long run.

Overall, the traces for each $w_i(t)$ during a single GradNorm run seem fairly stable and convergent. In fact, 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 an expanded version of the NYUv2 (Nathan Silberman & Fergus (2012)) dataset, with 40,000 images complete with pixel-wise depth, surface normals, and room keypoint labels. Keypoint labels are obtained through professional labeling, while surface normals are generated from camera parameters and the depth maps through standard methods. As it is an unofficial dataset, expanded NYUv2 gives noisier labels than the normal NYUv2, but compensates with an almost 100-fold increase in training data. Inputs are downsampled to 320 x 320 pixels and outputs to 80 x 80 pixels.

We also test GradNorm on the standard NYUv2 dataset, which carries depth, surface normals, and semantic segmentation labels (which we cluster into 13 distinct classes). This dataset is much smaller, with a training split of $\sim$800 examples, and we also downsample this dataset to the same 320 x 320 input with 80 x 80 output. This downsampled NYUv2 dataset allows us to demonstrate that our methods work with a cross-entropy classification (segmentation) task added to the mix.

### 5.1 Model and Individual Task Losses

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 to maximize 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, our VGG model has $\sim$29 million parameters to our thin ResNet's $\sim$14 million. By designing two extremely different network topologies, we will demonstrate that GradNorm is very robust to the choice of base model.

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 our regression losses are quadratic (for normals, $1-|\cos\theta| \approx \theta^2+\mathcal{O}(\theta^4)$), but we mix in cross-entropy loss to show the robustness of GradNorm across various loss functions. In Section 5.2 we also discuss how to generalize our methods to other loss functions.

### 5.2 Network Performance

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

Table 1: **Test error, expanded NYUv2 for GradNorm and various baselines.**

In Tables 1 and 2 we display the performance of our networks on the expanded and downsampled NYUv2 datasets, respectively. In both cases, GradNorm networks outperform other multitask methods, and either matches (within noise) or surpasses the performance of single-task networks.

| **Model Type and Weighting Method** | **Depth Error (m)** | **Segmentation 1-mIoU (%)** | **Normals Error (1-\|cos\|)** |
|---|---|---|---|
| VGG SegNet, Depth Only | 1.038 | - | - |
| VGG SegNet, Keypoint Only | - | 70.0 | - |
| VGG SegNet, Normals Only | - | - | **0.169** |
| VGG SegNet, Equal Weights | 0.944 | 70.1 | 0.192 |
| VGG SegNet, GradNorm Converged Weights | 0.939 | **67.5** | 0.171 |
| VGG SegNet, GradNorm $\alpha = 1.5$ | **0.925** | 67.8 | 0.174 |

Table 2: **Test error, downsampled NYUv2 for GradNorm and various baselines.**

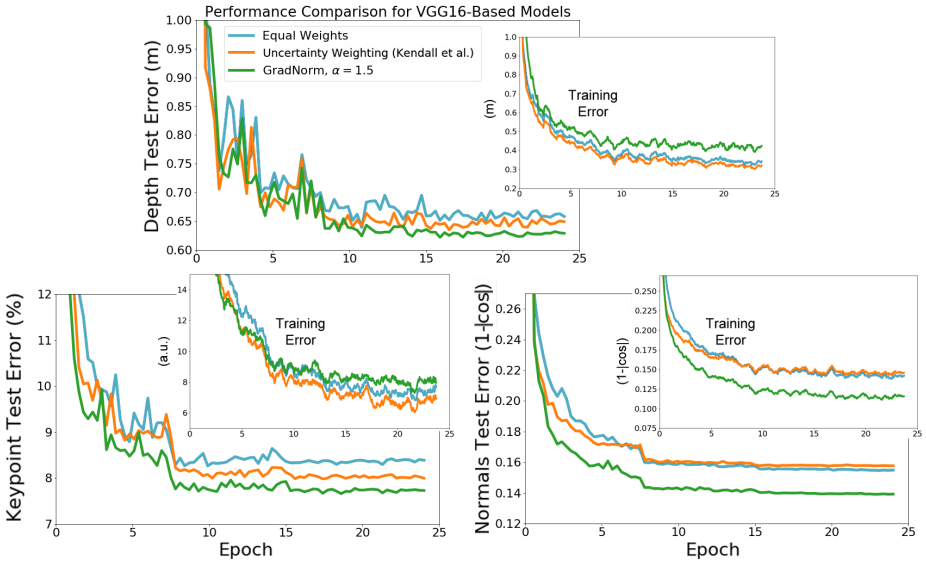


Figure 3: **Test and training loss curves for all tasks in expanded NYUv2, VGG16 backbone**. GradNorm versus an equal weights baseline and uncertainty weighting (Kendall et al. (2017)).

In Tables 1 and 2, we also find that the GradNorm Converged Weights network performs quite well. This network is derived by calculating the GradNorm time-averaged weights $E_t[w_i]$ for each task (e.g. by averaging the curves in Figure 5), 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 the Section 5.3 and show that these weights lie very close to the optimal weights extracted from exhaustive grid search.

It may be surprising that GradNorm can balance regression and classification losses in the same network. Some theoretical insight for why this might be the case can be found in Kendall et al. (2017), where the authors derive that (under some mild assumptions) the estimated intrinsic task uncertainty $\sigma_i$ follows the same rule $\sigma_i \sim \sqrt{L_i}$ for both cross entropy and squared losses.

In the case where we use a loss function that is a transform $\phi$ of either cross-entropy or squared error (for example, an $L_1$ loss has $\phi(L) = \sqrt{L}$ with respect to squared error), then the loss ratio needs to be updated to $L' = \phi^{-1}(L/L^{(0)})$ so that the learning rates $L'$ are all on an appropriate shared scale. Mixing in more exotic loss functions may require additional statistical reasoning to find the right form of $L'$, but cross entropy and squared error (and functional transforms thereof) cover most loss functions used in practice, and so GradNorm can be easily adapted to various scenarios.

Our optimal GradNorm network corresponds to $\alpha = 1.5$, revealing a significant asymmetry in learning dynamics between the different tasks in NYUv2. GradNorm handles the asymmetry by regularizing the network and reining in the training rates of overfitting tasks. Figure 3 displays how GradNorm improves test-time depth error on the expanded NYUv2 dataset by $\sim 5\%$, even though its depth training loss is worse. The same trend exists for keypoint regression, and is a clear signal of network regularization. In contrast, the uncertainty weighting technique (Kendall et al. (2017)) causes both test and training error to move in lockstep, 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 expanded NYUv2. 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 $\alpha = 1.5$ VGG network at the same 15000 steps. The results are shown in Figure 4.

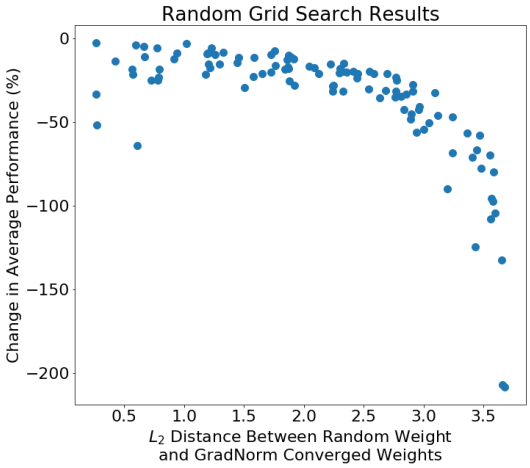


Figure 4: **Gridsearch performance for random task weights, expanded NYUv2**. Average change in performance across three tasks for a static network with weights $w_i^{\text{static}}$ is plotted against the $L_2$ distance between $w_i^{\text{static}}$ and our GradNorm network's time-averaged weights, $E_t[w_i(t)]$. All comparisons are made at 15000 steps of training.

Even after 100 networks trained, grid search still falls short of our GradNorm network. But even more remarkably, there is a strong, negative correlation between network performance and task weight distance to our time-averaged GradNorm weights. At an $L_2$ distance of $\sim 3$, grid search networks on average have almost double the errors per task compared to our GradNorm network. GradNorm has effectively allowed us to "cheat" and immediately find the optimal grid search weights without actually performing grid search, simplifying a process that is usually notoriously laborious.

### 5.4 Effects of tuning the asymmetry $\alpha$

The only hyperparameter in our technique is the asymmetry $\alpha$. The optimal value of $\alpha$ 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 why we call $\alpha$ an asymmetry parameter.

Tuning $\alpha$ 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. Figure 5 shows that higher values of $\alpha$ 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.

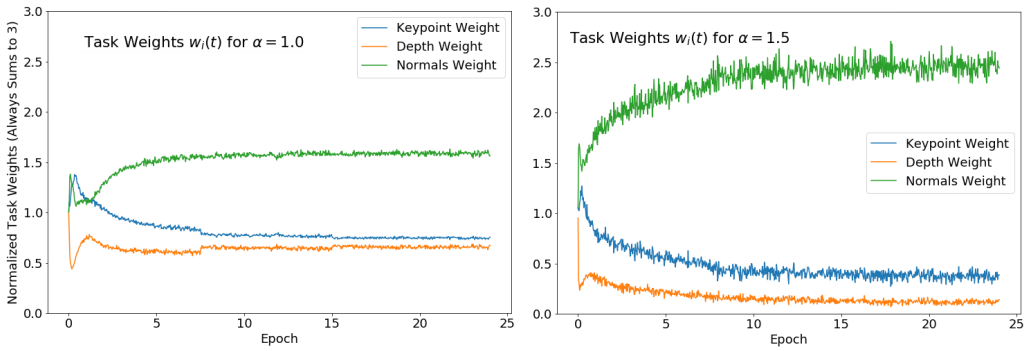


Figure 5: **Weights $w_i(t)$ during training, expanded NYUv2.** Traces of how the task weights $w_i(t)$ change during training for two different values of $\alpha$. A larger value of $\alpha$ pushes weights farther apart, leading to less symmetry between tasks.

### 5.5 General Training Characteristics

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. NYUv2 runs begin with a learning rate of 2e-5. Expanded NYUv2 runs last 80000 steps with a learning rate decay of 0.2 every 25000 steps. Downsampled NYUv2 runs last 20000 steps with a learning rate decay of 0.2 every 6000 steps. Updating $w_i(t)$ is done 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.

### 5.6 Qualitative Results

Figure 6 shows visualizations of the VGG SegNet outputs on test set images along with the ground truth, for both the expanded and downsampled NYUv2 datasets. The improvements are incremental, but the GradNorm network consistently outputs smoother, more detailed pixel map predictions.

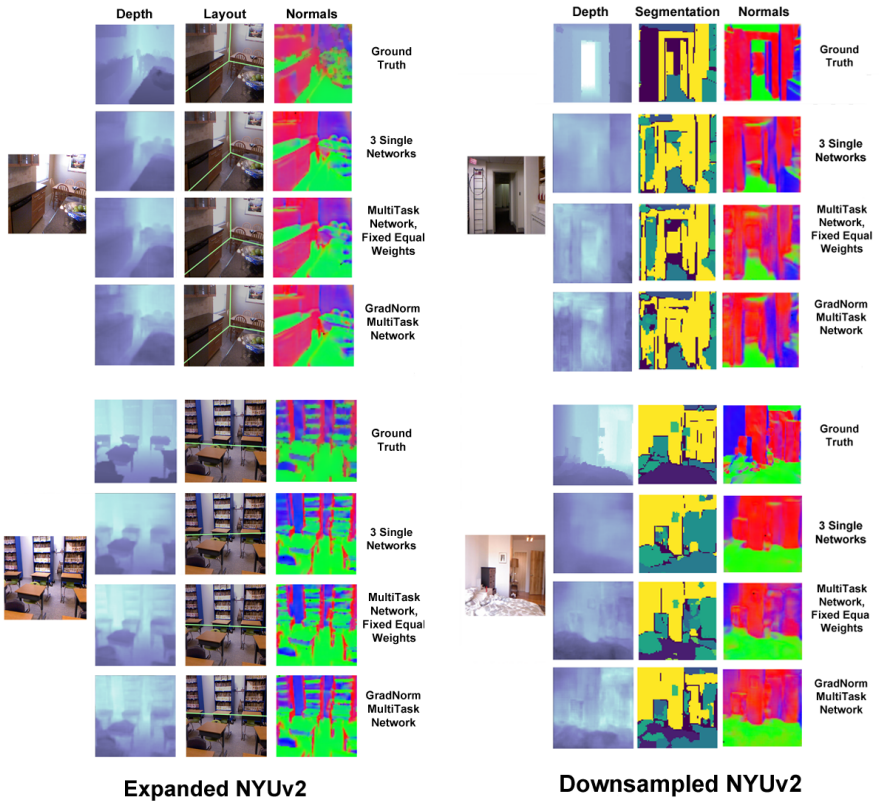


Figure 6: **Visualizations at inference time.** Expanded NYUv2 with room layout labels is shown on the left, while downsampled NYUv2 with semantic segmentation labels is shown on the right.

## 6 Conclusions

Gradient normalization acts as a good model regularizer and leads to superb performance in multitask networks by operating directly on the gradients in the network. GradNorm operates under an attractively simple heuristic of rate balancing, and can accommodate problems of varying complexities within the same unified model using a single hyperparameter representing task asymmetry. A GradNorm network also can be used to quickly extract optimal fixed task weights, removing the need for exhaustive grid search methods that become exponentially more expensive with the number of tasks. We hope that our work has not only introduced a new methodology for quickly balancing multitask networks, but also has shown how direct gradient manipulation can be a powerful way to reason about task relationships within a multitask framework.

## REFERENCES


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

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

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

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

Ronan Collobert and Jason Weston. 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.

David Eigen and Rob Fergus. 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.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

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

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

Laurent Jacob, Jean-philippe Vert, and Francis R Bach. Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems*, pp. 745–752, 2009.

Zhuoliang Kang, Kristen Grauman, and Fei Sha. 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.

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

Iasonas Kokkinos. 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.

Chen-Yu Lee, Vijay Badrinarayanan, Tomasz Malisiewicz, and Andrew Rabinovich. Roomnet: End-to-end room layout estimation. *arXiv preprint arXiv:1703.06241*, 2017.

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

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

Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *arXiv preprint arXiv:1611.05377*, 2016.

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

Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

Michael L Seltzer and Jasha Droppo. 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.

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

Anders Søgaard and Yoav Goldberg. 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.

Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. *arXiv preprint arXiv:1612.07695*, 2016.

Zhizheng Wu, Cassia Valentini-Botinhao, Oliver Watts, and Simon King. 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.