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

Gelareh Najmi
Department of Civil and Environmental Engineering
Old Dominion University
gnajm001@odu.edu
Dániel Bence Papp
Department of Computer Science
Old Dominion University
dpapp001@odu.edu

Abstract

Deep multitask networks, in which a single neural network provides many predicted outputs, can outperform single-task networks in terms of speed and performance, but they are difficult to train effectively. We introduce a gradient normalization (GradNorm) approach that dynamically tunes gradient magnitudes to automatically balance training in deep multitask models. GradNorm increases accuracy and decreases overfitting across multiple tasks when compared to single-task networks, static baselines, and other adaptive multitask loss balancing strategies for various network designs, regression and classification tasks, and synthetic and real datasets. Despite just needing a single asymmetry hyperparameter α, GradNorm meets or exceeds the performance of exhaustive grid search approaches. As a result, what was formerly a time-consuming search procedure that required exponentially more compute with each additional work may now be completed in a few training cycles, regardless of the number of tasks. Finally, we will show that gradient modification gives us a lot of control over the training dynamics of multitasking networks, and that it might be one of the keys to unlocking multitask learning's full potential.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

1 Introduction

In deep learning, single-task learning in computer vision has had a lot of success, with many single-task models currently performing at or above human accuracies for a wide range of tasks. However, an ultimate visual system for comprehensive scene awareness must be able to do many different perceptual tasks at the same time and quickly, especially in embedded systems like smartphones, wearable devices, and robots/drones, which have restricted computational environments. Multitask learning, in which one model distributes weights across many tasks and produces numerous inferences in one forward pass, can allow such a system (see Fig. 1). Not only are such networks scalable,

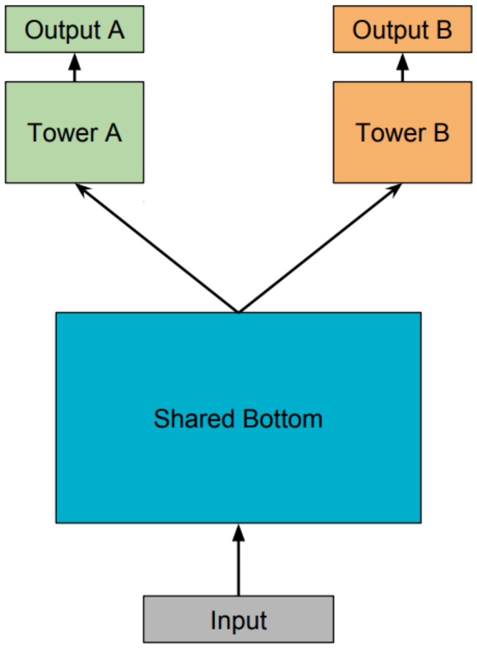

Figure 1: Multitask learning model: Shared-Bottom model

but the shared features within them can also induce more robust regularization and, as a result, improve performance. Multitask networks are challenging to train in general; diverse tasks must be carefully balanced such that network parameters converge to strong common characteristics that are applicable to all tasks. Methods in multitask learning have largely attempted to achieve this balance by manipulating the forward pass of the network (e.g., by constructing explicit statistical relationships between features [1] or optimizing multitask network architectures [2], but such approaches overlook a key insight: task imbalances obstruct proper training because they manifest as imbalances between backpropagated gradients. During training, for example, a task that is overly dominating will inevitably show its dominance by creating gradients with relatively large magnitudes. By directly altering gradient magnitudes via multitask loss function adjustment, we hope to address such difficulties at their source. In practice, in single task losses L_i, $L = \sum w_i L_i$, where the sum runs across all T tasks, the multitask loss function is frequently believed to be linear. In our example, we suggest an adaptive technique, which allows us to modify t: $w_i = w_i(t)$ at each training phase (t). Because the backpropagated gradient magnitudes from each job are extremely directly and linearly connected to this linear version of the loss function, it is handy for performing gradient balancing. The difficulty then becomes determining the ideal value for each wi at each training step t that balances each task's contribution for effective model training. We present a simple approach that penalizes the network when backpropagated gradients from any job are too high or too little in order to maximize the weights $w_i(t)$ for gradient balancing. When tasks train at equal rates, the right balance is reached; if task i is training fast, its weight $w_i(t)$ should drop in comparison to other task weights $w_j(t)|_{j \neq i}$ to give other tasks greater effect on training. Our approach is similar to batch normalization [3], but with two key differences: (1) we normalize across jobs rather than across data batches, and (2) we utilize rate balance as a desirable goal to guide our normalization. We'll illustrate how gradient normalization (hereinafter referred to as GradNorm) improves network performance while reducing overfitting dramatically. We'll illustrate how gradient normalization (hereinafter referred to as GradNorm) improves network performance while reducing overfitting dramatically.

1.1 Literature review

Multitask learning was developed long before deep learning [4, 5], but deep networks' strong learnt features and good single-task performance have reignited interest. Although our primary application area is computer vision, multitask learning has applications in a variety of other fields, including natural language processing [6, 7] speech synthesis [8], and traffic prediction [9]. Multitask learning has previously been investigated in the context of curriculum learning [10], where subsets of tasks are then learned based on local incentives; we look at the opposite approach here, where tasks are simultaneously trained based on global rewards such total loss reduction. Multitask learning is ideally suited to the field of computer vision, as producing numerous reliable predictions is critical for a thorough comprehension of a scene. Deep networks have been utilized to handle a variety of subsets of multiple vision problems, ranging from three-task networks [11] to far larger subsets [12]. Single computer vision tasks are frequently phrased as multitask problems, such as segmentation in Mask R-CNN [13] or object identification in YOLO-9000 [14]. The extensive and large amount of work on explicitly exploiting task interactions within a multitask paradigm is particularly noteworthy. Beyond deep models, clustering methods have shown success [15], while deep relationship networks [1] and cross-stitch networks [2] give deep networks the ability to search for meaningful relationships between tasks and to learn which features to share between them. Researchers in [16] and [17] used label groups to search for possible learning architectures. Kendall et al. [18] employs a joint likelihood formulation to estimate task weights based on inherent uncertainty in each task, which is perhaps the most relevant to the current study.

2 The GradNorm Algorithm

We want to learn the functions $w_i(t)$ for a multitask loss function $L(t) = \sum w_i L_i$ with the following goals: (1) to put multiple task gradient norms on a common scale so we may reason about their relative magnitudes, and (2) to dynamically modify gradient norms so that different tasks train at similar rates. To do this, we must first define the necessary quantities, first in terms of the gradients we will be modifying.

- W: The weights of a subset of the whole network. To economize on computation expenses, W is usually used as the last common layer of weights.
- $G_w^{(i)}(t) = \|\nabla_W w_i(t) L_i(t)\|_2$: with regard to the given weights W, the L_2 norm of the gradient of the weighted single-task loss $w_i(t)L_i(t)$.
- $\bar{G}_w(t) = E_{task}[G_w^{(i)}(t)]$: At training period t, the average gradient norm over all tasks.

We also establish different training rates for each task i:

- $\hat{L}_i(t) = \frac{L_i(t)}{L_i(0)}$: the loss ratio for task i at time t.
- $r_i(t) = \frac{\hat{L}_i(t)}{E_{task}[\hat{L}_i(t)]}$: the relative inverse training rate of task i.

We can now finish our explanation of the GradNorm algorithm with the following definitions in place.

3 GradNorm for Gradient Balancing

GradNorm should create a standard scale for gradient magnitudes, as well as balance training rates for distinct jobs, as described in the preceding Section. The average gradient norm, $\bar{G}_w(t)$, is the most popular scale for gradients. It creates a baseline at each timestep t by which we may calculate relative gradient sizes. To rate balance our gradients, we may utilize the relative inverse training rate of task i. To put it another way, the higher the value of $r_i(t)$, the greater the gradient magnitudes for task i should be in order to encourage the task to train faster. As a result, for each job i our desired gradient norm is simply:

$$G_w^{(i)}(t) \longrightarrow \bar{G}_w(t)[r_i(t)]^\alpha, \tag{1}$$

where α is a hyperparameter that has been added. We update our loss weights $w_i(t)$ to shift gradient norms towards this objective for each task using equation 1, which provides us a target for each task's gradient norms. GradNorm is then implemented as an L_1 loss function L_{grad} that sums the actual and goal gradient norms at each timestep for each task:

$$L_{grad}(t; w_i(t)) = \sum_i \| G_w^{(i)}(t) - \bar{G}_w(t)[r_i(t)]^\alpha \|_1, \tag{2}$$

where the total is applied to all T jobs.

4 Synthetic Data Generation

We create two regression tasks, inspired by Kang et al. [19], and utilize the Pearson correlation of the labels of these two tasks as a quantitative measure of task links. We set the regression model as a mixture of sinusoidal functions as used in [20], rather than the linear functions used in [19], because we are focusing on DNN models. In particular, we create synthetic data as follows:

1. We produce two orthogonal unit vectors u1,u2 from the input feature dimension d.

$$u_1{}^T u_2 = 0, \|u_1\|_2 = \|u_2\|_2 = 1 \tag{3}$$

2. Create two weight vectors w_1, w_2 with a scale constant c and a correlation value $0 \leq p \leq 1$ such that

$$w_1 = cu_1, w_2 = c\left(pu_1 + \sqrt{(1-p^2)u_2}\right) \tag{4}$$

3. Sample each element of an input data point x at random from $N(0, 1)$.
4. Create two labels, y_1 and y_2, for the following two regression tasks:

$$y_1 = w_1{}^T x + \sum_{i=1}^{m} sin\left(\alpha_i w_1{}^T x + \beta_i\right) + \epsilon_1$$

$$y_2 = w_2{}^T x + \sum_{i=1}^{m} sin\left(\alpha_i w_2{}^T x + \beta_i\right) + \epsilon_2, \tag{5}$$

where α_i, and β_i are given parameters that control the shape of the sinusoidal functions.

5 Result

To evaluate our Grad Norm algorithm, we used a combination of Weighted Mean Square Error and r^2 scores, which was plotted using matplotlib (see Fig. 2). We observed a consistent improvement over each iteration in the r^2 score across both tasks, achieving a high of 95%. When looking at wMSE scores which is a product of w_i and $\sum_{i=1}^{n}(x_i - y_i)^2$, we can see that over each iteration we were able to reduce the deviation of the predicted value from the target value. This is due to the design of Grad Norm, where we are fine tuning the weights of the loss function over each iteration. The weight coefficient is calculated using $c = \frac{2}{(w_1+w_2)}$ where w_1 is the previous weight of the first task and w_2 is the previous weight of the second task. This way we update the weights to be $c * w_1$ and $c * w_2$ which directly leads into the loss functions. We can see how the weights changed over the 50 epochs in (Fig. 3), where the sum of the weights always equals 3.

When looking at Figure 2, the difference between the training errors and testing errors cannot be overlooked. While running the algorithm using $\alpha = 0.15$, we weren't able to reduce the deviation between the testing and training errors. While in the original Grad Norm paper the researchers ran the algorithm through many more iterations, this wasn't quite possible as neither mine or research partner's computer is strong enough for it. When adjusting the α value, between the ranges $0 < \alpha >$ 2, significant improvements couldn't be seen. Some major improvements could be seen when we adjusted the batch sizes and the number of overall batches created by the algorithm but the exponential uptick in computation time rendered the adjustment of that parameter useless. We found that the

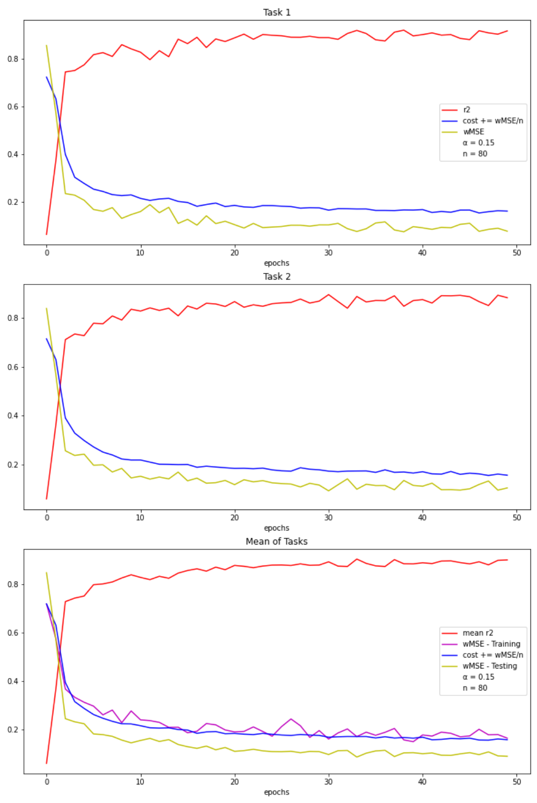

Figure 2: Evaluation of the Grad Norm algorithms using r^2, and Weighted MSE scores.

sweet spot when it comes to computation time and accuracy is a batch size of 100 and a learning rate of 0.001. This learning rate was shared by both the optimizer algorithms that are used in Grad Norm. Throughout the visualizations we can see a cost value associated with each task. This represented the mean cost of an epoch. The formula that was used to calculate the cost is $e_n = e_p + \frac{wSSE_l}{n}$ where e_p is the cost of the previous epoch, or 0 in case of the first epoch. The variable $wSSE_l$ is the weighted loss function of task l and n is the number of mini-batches. The number of mini-batches depends on the input size of the data-frame and it's denominator, mb which was a constant 100 throughout the execution of Grad Norm.

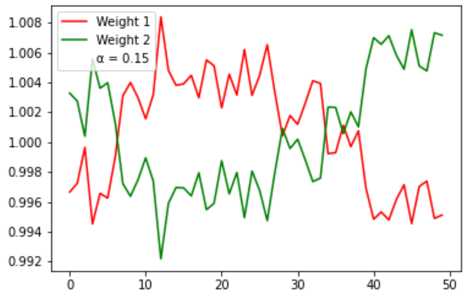

Figure 3: The weights of the loss functions changing over 50 epochs, but always equalling 3.

References

[1] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S Yu. Learning multiple tasks with multilinear relationship networks. *Advances in neural information processing systems*, 30,

2017.

[2] 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*, pages 3994–4003, 2016.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[4] Rich Caruana and Joseph O'Sullivan. Multitask pattern recognition for vision-based autonomous robots. In *International Conference on Artificial Neural Networks*, pages 1115–1120. Springer, 1998.

[5] BJ Bakker and TM Heskes. Task clustering and gating for bayesian multitask learning. 2003.

[6] 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*, pages 160–167, 2008.

[7] 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.

[8] Michael L Seltzer and Jasha Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6965–6969. IEEE, 2013.

[9] 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.

[10] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017.

[11] 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*, pages 2650–2658, 2015.

[12] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6129–6138, 2017.

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[14] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[15] Laurent Jacob, Jean-philippe Vert, and Francis Bach. Clustered multi-task learning: A convex formulation. *Advances in neural information processing systems*, 21, 2008.

[16] David Warde-Farley, Andrew Rabinovich, and Dragomir Anguelov. Self-informed neural network structure learning. *arXiv preprint arXiv:1412.6563*, 2014.

[17] 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. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5334–5343, 2017.

[18] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[19] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011.

[20] Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning structured weight uncertainty in bayesian neural networks. In *Artificial Intelligence and Statistics*, pages 1283–1292. PMLR, 2017.