ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

Evolutionary Deep Multi-Task Learning

Patrick Burke, Jonas Prellberg, and Oliver Kramer

University of Oldenburg
Department of Computer Science
26122 Oldenburg, Germany

Abstract. Multi-task learning is an approach to reduce the amount of required training data by learning multiple tasks at the same time. In the context of neural networks, multi-task learning is performed by sharing weights or creating dependencies between weights of task-specific networks. In this work, we propose an algorithm that uses a simple evolutionary algorithm, which is able to match and also surpass learned weight sharing. We evaluate the performance of this method on CIFAR-100, cast as a multi-tasking problem, using an 18-layer residual network, and compare our results to literature.

1 Introduction

Humans are excellent at learning from few examples, while deep neural networks struggle in this setting. In the context of neural networks, *Multi-task learning* (MTL) [1] is performed by sharing weights or creating dependencies between weights of task-specific networks. The idea is to put pressure on the weights to perform well under different tasks, which improves generalization. MTL can be employed to improve the test error in some or all tasks compared to single-task learning (STL), or to increase efficiency by matching the STL performance while reducing the total number of weights.

We focus on MTL with hard parameter sharing between task-specific neural networks. It is common to share weights between corresponding layers of the networks, but it is difficult to decide which tasks should share weights with which other tasks, i.e., to decide on a *weight sharing scheme*.

As a general rule, it makes sense to share early layers in convolutional neural networks between many tasks because features are general [2] and applicable to many tasks. However, for layers closer to the network output it is unclear which tasks would benefit one another. Finding a weight sharing scheme is a combinatorial problem that depends on the number of tasks, layers, and weights. Consequently, the search space can easily become very large and finding a well-performing scheme by hand requires tedious and error-prone experimentation. Furthermore, every experiment requires to train neural networks, which is a time-consuming step.

Several algorithmic approaches to solving MTL problems have been proposed, a more recent example by Prellberg and Kramer [3] called *learned weight sharing* (LWS) combines stochastic gradient descent and natural evolution strategies to simultaneously optimize weights and a weight sharing scheme.

Section 2 presents related work. Our method is introduced in Section 3 and experimentally analyzed in Section 4. Conclusions are drawn in Section 5.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

2 Related work

LWS forms the conceptual basis of our method. Weight modules are shared between layers at equal depth between task-specific networks of the same but arbitrary architecture. A probability distribution over assignments is optimized by a natural evolution strategy, while the weights themselves are trained by gradient descent in alternating steps.

The routing net algorithm [4] uses a small router network to dynamically select weight modules dependent on input data and task identifier. In contrast to our method, the router is trained by reinforcement learning and the weight modules are selected on a per-sample basis, whereas our architectures are fixed after the optimization is complete.

Cross-stitch networks [5] and *sluice networks* [6] combine the output of task specific layers using a learnable linear combination, which is then fed into the next layers. *Soft layer ordering* [7] extends this technique so that shared layers can be applied at different depths. Due to the weighted combination of outputs, all three of these methods require all networks during inference, even if only one task is being solved. In contrast, our approach produces task-specific networks that can be independently used for inference.

3 Evolved Weight Sharing

This paper proposes evolved weight sharing (EWS), which is visualized in Figure 1. The solutions are assignments between weights and layers in task-specific networks. For each layer in a network, we create a set of K weights that EWS can plug into that layer in a task-specific network. This non-differentiable assignment is optimized by an EA, while the differentiable weights themselves are optimized with standard methods such as Adam [8]. The fitness signal that drives the EA search process is the aggregated validation accuracy of all task-specific networks. Depending on this fitness signal, per-task mutation rates are dynamically adapted during runtime to push the search process towards better assignments.

The initial assignment is one without any weight sharing, i.e. each task-specific network has its own weights. EWS then starts by creating an offspring population of size λ through mutation of the solution with the highest fitness from the current population. This mutation changes only the assignment be-

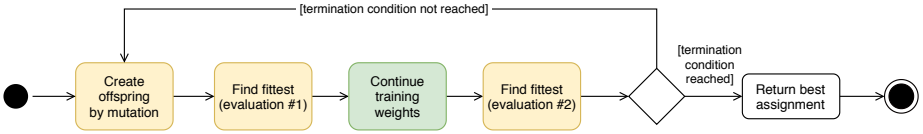

Fig. 1: Overview of the EWS algorithm. Activities are grouped by color into belonging to either assignment optimization or weight optimization.

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

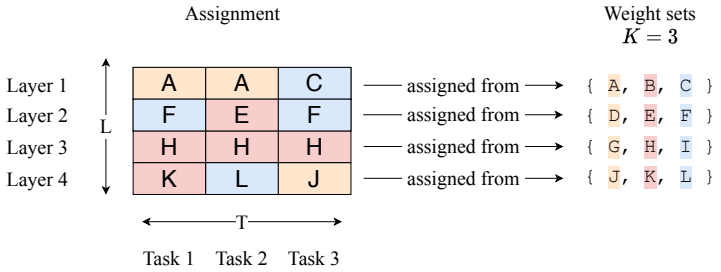

Fig. 2: Matrix-representation of an assignment for an architecture with five shareable layers and three tasks. Each element in the matrix refers to one of $K = 3$ weight modules, e.g. task 1 and 2 in layer 1 share weight module A, while task 3 has its own weight module C.

tween weights and task-specific layers but keeps the weight values unchanged. Each assignment in this offspring population can be represented by a matrix, as shown in Figure 2.

After the offspring population has been created, its fitness is evaluated. This is done because the assignments have changed through the mutation and we want to find the best-performing network under the new assignment. The fittest network's weights are then trained for a fixed number of steps, starting from its current weights. This weight inheritance allows the algorithm to train weights for many steps in total, even with a small number of training steps per generation. Furthermore, it is a form of transfer learning because the same weight will be trained on data from different tasks depending on the assignment. After training, the fitness is evaluated again to find the best performing solution to use as a parent.

3.1 Mutation

Random resetting is used as the mutation operator, with per-task mutation rates $\forall t \in T : m_t \in [0, 1]$. For every task t and layer l in an assignment α, a weight is kept unchanged with probability $1 - m_t$. With probability m_t, we assign to $\alpha_{l,t}$ a new weight chosen uniformly at random from the weight set Θ_l that corresponds to the layer l.

3.2 Mutation Rate Adaptation

The per-task mutation rates are adapted depending on the performance of the fittest assignment on every individual task. After each generation, the per-task performance difference Δp_t between the current fittest assignment and the h-th last fittest assignment is used to determine a new mutation rate m_t. In our experiments h is set to 3. Equation 1 shows how mutation rates are adapted

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

depending on the change in performance:

$$\forall t \in T : m_t = \begin{cases} \max(m_{min}, m_t - 0.05), & 0.1 \leq \Delta p_t \\ m_t, & 0 \leq \Delta p_t < 0.1 \\ \min(m_{max}, m_t + 0.01), & \Delta p_t < 0 \end{cases} \tag{1}$$

In general, the mutation rate is increased if performance decreases so that it becomes more likely to create very different weight sharing schemes. If the performance increases moderately, the algorithm shows fast progresses and the mutation rate is kept unchanged. After very significant performance increases, the mutation rate is reduced to prevent the configuration from being changed too drastically. Finally, to make sure the search process does not come to a complete halt or degenerates into a random search, the mutation rates are kept between thresholds of $m_{min} = 0.05$ and $m_{max} = 0.2$.

3.3 Fitness

The fitness of an assignment α is measured by the harmonic mean of test set accuracies $p_t(\alpha, \Theta)$ for each task t given all weights Θ. The harmonic mean was chosen compared to an arithmetic mean because it will be more strongly influenced by low-value outliers [9]. This prevents assignments that sacrifice a lot performance on a single task to improve the average.

4 Experiments

We test EWS on the CIFAR-100 [10] cast as an MTL problem with 20 tasks, using two different architectures selected from previous work. The first architecture is a custom convolutional network from [4], and the second is a ResNet18 from [3]. This choice allows us to compare EWS to routing networks [4], cross-stitch networks [5], and learned weight sharing [3].

As an additional evaluation metric, we propose the reduction quotient, which describes what percentage of weights is saved by an assignment compared to single-task learning. A value of $r(\alpha) = 1$ means that the assignment a shares as many weights as possible. Let L be the number of shareable layers, T be the number of tasks and $d(\alpha, l)$ denote the number of distinct weights assigned in the layer l of an assignment A, then the reduction quotient $r(\alpha)$ is defined as the mean of all layer-wise reduction quotients:

$$r(\alpha) = \frac{1}{L} \sum_{l=1}^{L} \frac{T - d(\alpha, l)}{T - 1}. \tag{2}$$

4.1 Convolutional network

The network architecture used in [4, 3] consists of four alternating convolutional and max-pooling layers, followed by three fully connected layers with 128 units, and finally another fully connected layer for each task output. Only the three

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

fully connected layers with 128 units are shareable between tasks in their setup, so EWS will only be applied to these layers as well.

EWS is configured to choose from $K = 20$ weights for every layer and create $\lambda = 5$ offspring every generation. The training step uses the Adam [8] optimizer on 1 epoch of data in batches of size 256 and learning rate 10^{-3}. If the fitness does not improve for 40 generations, EWS terminates.

Method	Test error [%]	Reduction quotient
Cross-stitch networks [4]	47	
Routing networks [4]	40	
Full sharing [3]	39.08 ± 0.36	1
No sharing [3]	36.50 ± 0.43	0
LWS [3]	37.43 ± 0.53	
EWS	**36.12** ± 0.42	0.36

Table 1: EWS results (10 runs) using the convolutional network on CIFAR-100.

Table 1 compares results to literature. We can see that EWS beats all baselines, especially LWS. The EWS runs receive on average 12.6 epochs worth of training (this varies due to the termination condition), which is reasonably close to the 10.2 epochs that are used for LWS training.

4.2 Residual network

LWS [3] is also demonstrated using a ResNet18 [11] to show that the method works on large-scale architectures. We follow their adaptions of the ResNet18 to small image sizes present in CIFAR-100 by removing the 3×3 max-pooling layer and changing the convolution strides so that downsampling only happens in the last three stages. Just like in [3], residual blocks will be shared as a unit and the last layer is kept task-specific. All hyperparameters are identical to the previous section.

Method	Test error [%]	Reduction quotient
Full sharing [3]	31.80 ± 0.44	1
No sharing [3]	32.53 ± 0.32	0
LWS [3]	30.84 ± 0.49	
EWS (up to gen. 90)	**30.68** ± 0.32	0.37
EWS (up to gen. 263; single run)	**25.28**	0.36

Table 2: EWS results (10 runs) using the ResNet18 on CIFAR-100.

Table 2 shows that EWS can reach by far the lowest test error of 25.28 % after 263 generations (allowing EWS to run until performance does not improve for 40 generations). In this run, all weights were trained an average of

ESANN 2021 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 6-8 October 2021, i6doc.com publ., ISBN 978287587082-7.
Available from http://www.i6doc.com/en/.

169.2 ± 15.34 epochs. This is approximately three times the number of epochs used for LWS [3]. For a fairer comparison, we consider only results up to generation 90. Up to this point, all weights were trained an average of 52.50 ± 11.58 epochs, which is on the same level as the theoretical $\sim$51 epochs used in [3]. With this restriction, our method slightly outperforms LWS.

5 Conclusion

We observe that low values for the amount of offspring λ are sufficient for the search process to be successful. This may show that even sharing a few weights in the right way increases generalization pressure enough to improve the outcome. That being said, our gains over simple full sharing baselines and the algorithms from literature (cross-stitch networks, routing net, and learned weight sharing) highlight the importance of selecting specific assignments. Impressively, training EWS to convergence using the ResNet18 resulted in an absolute 5 percent-point improvement over LWS despite a much simpler algorithm.

References

[1] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[2] 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[3] Jonas Prellberg and Oliver Kramer. Learned weight sharing for deep multi-task learning by natural evolution strategy and stochastic gradient descent. *arXiv preprint arXiv:2003.10159*, 2020.

[4] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

[5] 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.

[6] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning, 2017.

[7] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *arXiv preprint arXiv:1711.00108*, 2017.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Jasmin Komić. *Harmonic Mean*, pages 622–624. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.