
DYNAMIC NEURAL DIVERSIFICATION: PATH TO COMPUTATIONALLY SUSTAINABLE NEURAL NETWORKS

PAPER ACCEPTED TO ICANN 2021.

Alexander Kovalenko
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
kovalale@fit.cvut.cz

Pavel Kordík
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
pavel.kordik@fit.cvut.cz

Magda Friedjungová
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
magda.friedjungova@fit.cvut.cz

ABSTRACT

Small neural networks with a constrained number of trainable parameters, can be suitable resource-efficient candidates for many simple tasks, where now excessively large models are used. However, such models face several problems during the learning process, mainly due to the redundancy of the individual neurons, which results in sub-optimal accuracy or the need for additional training steps. Here, we explore the diversity of the neurons within the hidden layer during the learning process, and analyze how the diversity of the neurons affects predictions of the model. As following, we introduce several techniques to dynamically reinforce diversity between neurons during the training. These decorrelation techniques improve learning at early stages and occasionally help to overcome local minima faster. Additionally, we describe novel weight initialization method to obtain decorrelated, yet stochastic weight initialization for a fast and efficient neural network training. Decorrelated weight initialization in our case shows about 40% relative increase in test accuracy during the first 5 epochs.

Keywords Diversification · Negative Correlation · Weight Initialization · Computational Sustainability · Neural Networks

1 Introduction

Over the last decade, machine learning algorithms have achieved vast progress in various fields. Namely, general approach called deep neural networks (DNN) with multiple hidden layers [1], has enabled machine learning algorithms to perform at an acceptable level in the many areas, in some cases outperforming human accuracy [2]. Such progress, in no small measure, has become available due to modern hardware computational capabilities, enabling the training of large DNN on an immense amount of data.

On the other hand, even though large models perform very well on complex tasks, we cannot endlessly rely on an infinite increase in computational resources and size of datasets. Training large neural networks is energy, time and memory demanding task. Recently, researchers started questioning energy consumption of machine learning algorithms and their carbon footprint [3]. Thus it will not be superfluous to develop a strategy for the models that have a constrained number of parameters, sufficient enough for the certain task, and can be trained fast, rather than chasing higher accuracy by enlarging the number of parameters and using more complex hardware.

Universal approximation theorem [4] claims that a feed-forward artificial neural network with a single hidden layer can approximate any continuous well-behaved function of arbitrary number of variables with any accuracy. The conditions are: a sufficient number of neurons in the hidden layer, and a correct weight selection. Above mentioned theorem for an arbitrary width case was originally proved by Cybenko [4] and Hornik [5] and later extended to an arbitrary depth case (DNN) in [6].

In this paper we get a deeper insight on the practical application of Cybenko’s theorem, in order to train a neural network, where all hidden neurons will be used efficiently. Therefore, we have to pay attention to two following aspects: number of neurons and correct weight selection.

Number of neurons in a hidden layer is a quite straightforward parameter that became trendy with availability of multi-threaded parallel computing on GPU [7]. Models of a vast number of trainable parameters are not devoid of logic, as they generalize better and can be so-called ‘universal learners’. For example, GPT-3 having 175 billion parameters, is a perfect example of a universal learner [8]. Thus, the community has been experimenting with model architectures increasing width [6] or depth [9] of neural networks. Issues, such as vanishing gradient [10, 11] was resolved by applying methods, including second-order Hessian-free optimization [12], training schedules by using greedy layer-wise training [13–15], sparse rectifier activation function, widely known as ReLU [16], layer-size-dependent initialization, such as Xavier [17] and Kaiming [18] and skip connections [19]. Even though, we can make arbitrarily large models make good predictions, to achieve computational sustainability by expanding the number of trainable parameters up to infinity, would not be the best option for the tasks of lower complexity. The community has been already trying to address this problem, thus several solutions dealing with this issue have occurred. For example, widely used ReLU activation function, saturated only in one dimension, which helps with vanishing gradient problem, on the other hand results in so-called ‘dying neurons’ [20], modified activation functions such as Leaky ReLU [21], adaptive convolutional ReLU [22], Swish [23], Antirectifier [24] and many other were addressed to solve the problem of ‘neural graveyard’. Resource efficient solutions, such as pooling operations [25], LightLayers [26] depth-wise separable convolutions [27] were developed to reduce the complexity of the models.

Correct weight selection, at first sight, depends on training parameters, such as loss function, number of epochs, learning rate etc. However, to train the neural network competently these weights have to be initialized stochastically. There are several ways to initialize weight, mainly aimed to avoid vanishing gradients. Nevertheless, stochastic weight initialization can result in neuron redundancy, when different neurons are trained in a similar manner. This is not crucial if the neural network is excessively large, however, in computationally sustainable models, neuron redundancy and ‘neural graveyards’ are undesirable. Moreover, there are numerous application when memory efficient model is required (e.g. autonomous devices such as sensors, detectors, mobile or portable devices). Such devices require memory and performance efficient solutions to learn spontaneously and improve from experience. In this case adding excessive parameters to the model can be rather questionable for the model application.

Therefore, once we consider each neuron of the model as an individual learner, the neural network can be seen as an ensemble. It is known that for ensembles diversity of learners is desirable to some extent [28]. Thus, we can assume that diversity between neurons or reinforced diversification during the training can be beneficial for the model.

In this paper we foremost explore how the diversity between neurons evolves during the training and as a following step suggest methods for diversification of the neurons during the model training. This is especially relevant in resource constrained models, where neuron redundancy means reducing the number of predictors. Additionally, we show how weight pre-initialization can affect neural network training at the early steps.

2 Our Approach

Let us start with a term *negative correlation* (NC) learning [28], which is a simple, yet elegant technique to diversify individual base-models in the ensemble and reduce their correlations. Ambiguity decomposition [29] of the loss function raises the possibility of controlling the trade-off between bias, variance, and covariance [30] using the strength parameter, to reduce covariance. In its order the concept of an NC learning is originated from bias-variance decomposition [31, 32] of ensemble learning. In this case, bias is the output shift from the true value, and variance is the measure of ensemble ambiguity, which simply means dispersion around the mean output value.

As it was first demonstrated by Krogh and Vedelsby [33] quadratic error of ensemble prediction is always less than the quadratic error of each individual estimator of the ensemble:

$$(f_{\text{ens}} - d)^2 = \sum_i w_i (f_i - d)^2 - \sum_i w_i (f_i - f_{\text{ens}})^2 \quad (1)$$

Later Brown [28] demonstrated decomposition of ensemble error into three components - bias, variance and covariance, and shown, the connection between ambiguity and covariance:

$$E \left\{ \frac{1}{M} \sum_i (f_i - d)^2 - \frac{1}{M} \sum_i (f_i - \bar{f})^2 \right\} = \overline{bias}^2 + \frac{1}{M} \overline{var} + \left(1 - \frac{1}{M}\right) \overline{covar} \quad (2)$$

The ensemble ambiguity is nothing less than the variance of the weighted ensemble around the weighted mean. Therefore, higher ambiguity, i.e. decorrelation between the ensemble output is desirable up to some measure.

Our *first trial* was to decorrelate neurons in the hidden layer by penalizing the difference between mean weight of the neurons \bar{w} and each neuron w_i :

$$NC = \frac{1}{n} \gamma \sum_i (\bar{w} - w_i) \quad (3)$$

where γ is the regularization strength parameter, and n is the number of neurons in a layer.

However, it is likely more profitable to compare not only single weights, but weight matrices or e.g. kernels in convolutional neural networks (CNN), as trainable kernels represent. Thus, the *second* way to define diversity is comparing neurons by cosine similarity:

$$\frac{1}{D} = \frac{1}{n} \gamma \sum_i \sum_j w_i \cdot w_j \quad (4)$$

where w are weights of individual neurons and D is the diversity measure.

In this technique we compare each weight in the layers and define a diversity measure D . However, it has quadratic complexity of such expression, which would oppose the idea of the current work, as our indent is fast and efficient training of resource constrained neural networks.

Therefore, combining the first two approaches we introduce and explore *another method* to define diversity in the neural networks:

$$\frac{1}{D} = \frac{1}{n} \gamma \sum_i \bar{w} \cdot w_i \quad (5)$$

After observing the training process and evolution of diversity measure in the models, we explored the possibility of weight pre-optimization using diversification. In this case, we used Kaiming weight initialization, with further optimization to enlarge the diversity between the weights, and at the same time keep weight mean and standard deviation of the weight matrix close to initial:

$$L = (|\bar{W} - \bar{w}_k| + |\sigma_w - \sigma_{w_i}|) \sum_i \sum_j w_i \cdot w_j \quad (6)$$

where L is loss, \bar{W} is the initial weight mean, \bar{w}_k is the weight mean at k training step, σ_w is standard deviation of the initial weights array, and σ_{w_i} is standard deviation of the weights array at k training step.

3 Experiments

We perform some initial experiments using DNN in order to study diversity evolution during the model training and demonstrate the effectiveness of proposed diversification mechanisms.

The experiments were performed on publicly available benchmark dataset Fashion MNIST [34]. This dataset was chosen as it is suitable for DNN training and has higher variance than traditional hand-written digits dataset MNIST [35]. We implemented one-hidden-layer neural network with 16, 32, 64, 128, and 256 neurons in the hidden layer (see Table

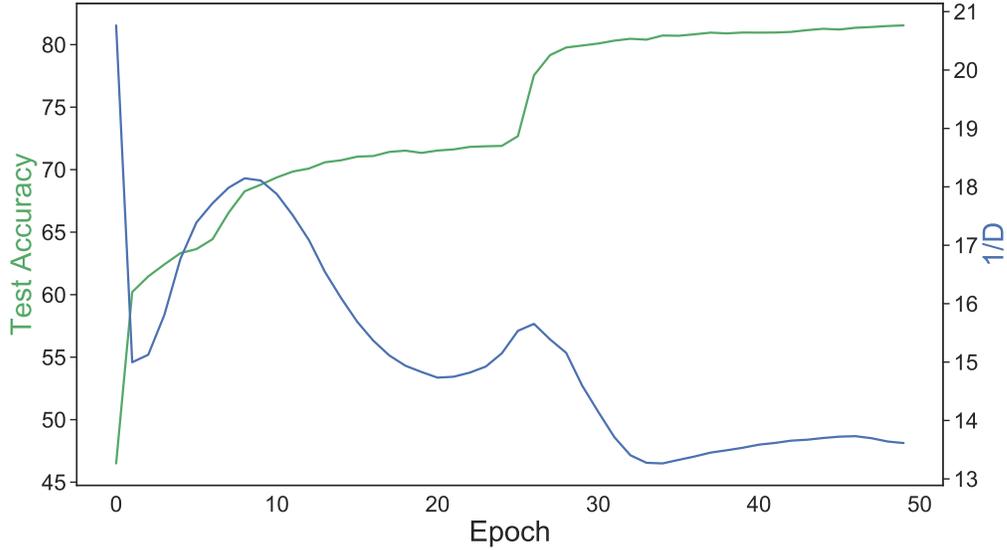


Figure 1: Training curve and Diversity measure (Eq. 4) for the first 50 epochs on Fashion MNIST dataset. DNN with 1 hidden layer of 32 neurons.

1), using PyTorch [36] library. Otherwise, we used standard parameters for the training, including Adam optimizer [37] with a learning rate of 0.01, cross entropy loss function with penalization terms (Eq. 3-5):

$$H(T, p) = - \sum_{i=1}^N \frac{1}{N} \log_2 q(x_i) + \frac{1}{D} \quad (7)$$

where T presents training set, p is true distribution, q is predicted distribution, N is standard deviation of the weights array at k training step, $q(x)$ is the probability of event x estimated from the training set, and D is the diversity measure, obtained using Eq. 3, 4, or 5.

4 Results and Discussion

4.1 Evolving Diversity and Symmetry Breaking

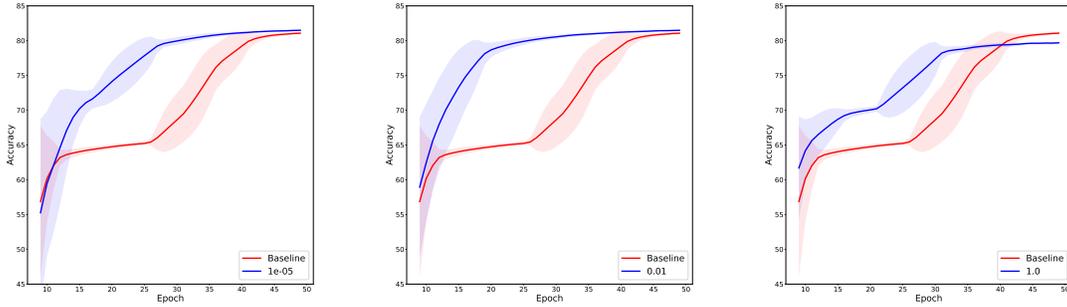
During the model training, one can notice sub-optimal accuracy stagnation for a several epochs, this can be associated with the existence of local minima on a loss function surface [38, 39]. This can be associated with a symmetry in the neural network layer, which is shown to be a critical point especially for small neural networks [40, 41]. We found out that naturally the model tends to decrease the correlation between the neurons, however when the model converges to a local minimum with a sub-optimal accuracy, the similarity between the neurons rises up until the moment when the optimization process surpasses the local minimum and the accuracy increases. (see Figure 1) This correlates with an existence of symmetry in the weights, once weights are symmetrical (correlated) and the number of neurons is constrained, the overall output of the model will likely to be inefficient.

4.2 Negative Correlation Learning

The experiment above inspired us to study certain ways to decorrelate neurons in the hidden layer, thus brake the symmetry that can appear during the learning process. As we discussed earlier, we consider the output of neural network as an output of an ensemble. Thus, first, we did simple NC learning, applied to the individual neurons, rather than ensemble of classifiers. The logic behind this experiment was rather comprehensible. Once the model has constrained number of parameters to generalize the data, higher variance would help to eliminate redundant neurons and overall prediction has to be more accurate. As it can be seen from the Figure 2. decorrelation mechanism helps to avoid local

Table 1: First 10 epochs average of the neural network training for various number of neurons, hidden layer diversified according to the Eq. 5.

γ	Number of Neurons & Test Accuracy, %				
	16	32	64	128	256
0.0	54.19	58.25	62.46	69.62	72.10
$1 \cdot 10^{-5}$	55.17	60.17	62.45	68.64	70.65
$1 \cdot 10^{-4}$	56.41	61.25	64.13	70.32	72.27
$1 \cdot 10^{-3}$	54.48	60.81	65.04	70.45	72.83
$1 \cdot 10^{-2}$	53.54	60.04	63.19	70.26	72.36
$1 \cdot 10^{-1}$	54.22	59.46	62.23	70.20	71.64
1.0	50.09	57.49	60.53	69.84	71.65


 Figure 2: Validation accuracy training curves of the model with various various γ values.

minima at the early stage on the model learning. Nevertheless, decorrelation using NC learning generally did not result in the higher accuracy overall. We associate it to several factors, such as Kaiming weight initialization that help to avoid vanishing gradient, and Adam optimizer which is a replacement optimization algorithm that can handle sparse gradients on noisy data, and thus is able to efficiently overcome local minima due to adaptive learning rated for each parameter. Eventhough, these widely used techniques are dealing with the above mentioned problem of the neuron redundancy, our proposed model can help at the early stages of a model training.

Moreover, with an increasing number of neurons the influence of decorrelation diminishes, this can be explained, that excessively large NN performs good at the low variance data as well as not every neuron is needed for a good prediction. However, in the present work we consider computationally sustainable DNN, where all the neuron are forced to contribute the prediction and on the other hand, for complex data larger amount of neurons would be needed to generalize the dataset. Therefore, for more sophisticated problems neuron diversification may be efficient for a larger number of neurons. However, in the present case we performed further experiments on the model with 64 neurons in the hidden layer, which we consider sufficient for a given dataset. All the models were trained for 10 times to calculate mean and standard deviation. In Table 2 the average testing accuracy of the first 10 epoch for the DNN with 64 neurons in the hidden layer trained using negative correlation learning (Eq. 3) is shown.

Table 2: First 10 epochs average of the neural network training, hidden layer diversified according to the Eq. 3.

γ	Train Acc., %	Test Acc., % ,	Test Acc. STD
0.0	61.46	62.46	2.34
$1 \cdot 10^{-5}$	62.26	62.45	2.34
$1 \cdot 10^{-4}$	63.65	64.13	1.59
$1 \cdot 10^{-3}$	63.12	65.04	1.76
$1 \cdot 10^{-2}$	62.54	63.19	1.03
$1 \cdot 10^{-1}$	64.23	62.23	0.95
1.0	59.56	60.53	1.57

4.3 Pairwise Cosine Similarity Diversification

It has to be noted that, unlike in [28], where universal diversification strength parameter was found for the ensembles of all sizes, in our case γ value depends on the size of the hidden layer and has to be rather considered as γ *per neuron*. However, on the other hand it is loss-dependent, which means that, ideally, it has to be same or one order of magnitude smaller than the output of the loss function during the training, otherwise, rather than the model loss (e.g. cross entropy), reciprocal diversity measure $\frac{1}{D}$ will be optimized. Thus, the reader has to consider optimizing γ value for each certain neural network and loss function. Thus optimal γ approximately can be estimated as:

$$\gamma = \frac{0.5 \cdot 10^{b_{loss}}}{n} \quad (8)$$

where n is the number of neurons in the hidden layer and b_{loss} is the loss function order of magnitude.

In addition to NC learning, we introduced diversity measure based on cosine similarity between the neurons (Eq. 4). Such technique, seems to be promising due to several reasons: first, we, rather than mean values, compare patterns, which can be useful for more complex models, such as CNNs or transformers, moreover here, each neuron is compared with each, thus such model is intended to be more robust. Nevertheless, at least for DNN, results we comparable with NC learning (see Table 3), additionally, such method has quadratic complexity, which opposes our initial aim to train small models faster and more efficient.

Table 3: First 10 epochs average of the neural network training, hidden layer diversified according to the Eq. 4.

γ	Train Acc., %	Test Acc., % ,	Test Acc. STD
0.0	55.94	62.83	1.18
$5 \cdot 10^{-8}$	56.52	64.20	1.11
$5 \cdot 10^{-7}$	57.98	65.76	0.92
$5 \cdot 10^{-6}$	55.48	59.96	0.71
$5 \cdot 10^{-5}$	56.47	49.20	1.52
$5 \cdot 10^{-4}$	56.47	44.60	1.06
$5 \cdot 10^{-3}$	42.66	38.61	1.10

4.4 Reaching Linear Complexity

To enable our diversification method to compare patterns, however avoid quadratic complexity, we combined the first concept of NC learning with the second one, and implemented diversity measure based on penalization of the cosine similarity of each neuron in the hidden and layer’s neurons mean (Eq. 5). The algorithm (see Table 4) overhead is comparable with L regularization. Moreover, it has shown the highest accuracy gain among three.

Table 4: First 10 epochs average of the neural network training, hidden layer diversified according to the Eq. 5.

γ	Train Acc., %	Test Acc., % ,	Test Acc. STD
0.0	61.54	63.15	2.08
$5 \cdot 10^{-7}$	62.37	63.3	1.63
$5 \cdot 10^{-6}$	63.60	64.54	1.25
$5 \cdot 10^{-5}$	64.87	64.95	1.66
$5 \cdot 10^{-4}$	60.36	62.14	1.66
$5 \cdot 10^{-3}$	52.54	55.86	0.45
$5 \cdot 10^{-2}$	41.26	42.71	1.32

4.5 Iterative Diversified Weight Initialization

However, it can be noticed, that occasionally, during the training, the model do not behave exactly as expected, creating an outlying learning curves. This is most likely associated with stochastic weight initialization. In this case Kaiming initialization is used [19]. Kaiming initialization is widely used for the neural networks with ReLU activation functions and related to the nonlinearities of the ReLU activation function, which make it non-differentiable at $x = 0$. The weights, in this case are initialized stochastically with the variance that depends on the number of neurons N :

$$v^2 = 2/N \quad (9)$$

It is fair to suggest, that correlation between the initialized weights can play significant role in the model learning process. Indeed, in the Figure 1. it is clearly seen, the the model gained the most of its accuracy while reducing the correlation between neurons during the first few epochs. However, the aim of weight initialization is to prevent layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. Usually weight are initialized stochastically with a small number to avoid vanishing gradients especially if *tanh* or *sigmoid* activation functions are used. Thus, to obtain stochastically initialized, yet decorrelated, weights we introduced iteratively diversified Weight initialization, using custom loss function based on Eq. 6. The logic behind such initialization is to reduce the diversity measure between the weights and at the same time keep weights mean \bar{w} and weights standard deviation σ_w close to the originally initialized using Kaiming initialization.

Table 5: First 5 epochs average of the neural network training initialized with decorrelated weights according to the Eq. 6 pre-optimized for 5 epochs.

γ	Train Acc., %	Test Acc., % ,	Test Acc. STD
0.0	29.54	34.23	2.04
$1 \cdot 10^{-4}$	42.43	43.43	1.81
$1 \cdot 10^{-3}$	43.92	45.65	1.53
$1 \cdot 10^{-2}$	44.65	47.01	1.24
$1 \cdot 10^{-1}$	38.32	39.83	1.06
1.0	36.64	38.94	1.37
10.0	32.5	37.57	1.41

5 Conclusion

In this paper we show how to explore and tame the diversity of neurons in the hidden layer. We studied how the correlation between the neurons evolves during the training and what is the effect on prediction accuracy. In appears, that once the model is converged to the local minimum on the loss landscape, correlation between the neurons increases up to the point when the optimization process overcome the local minimum. Thus, we introduced three methods how to dynamically reinforce diversification and thus decorrelate neural network layer. The concept of negative correlation suggested by Brown [28] was reviewed and expanded. Instead of decorrelation individual neural networks in the ensemble we diversified neurons in the hidden layer, using three techniques: *negative correlation learning*, *cosine pairwise similarity*, *cosine similarity around the mean*.

First technique is originated from the neural networks ensembles and shows a decent performance in our example using DNN, however for more sophisticated models, such as CNNs and transformers, second and third technique is likely to be more advantageous as far as it can compare patterns. Additionally to reach correct weight selection, we introduced weight iterative optimization using weight diversification. It was shown that such techniques are suitable for the fast training of small models and notably affect their accuracy at the early stage. Which is a small, yet important step towards the development of a strategy towards energy-efficient training of neural networks.

Our future plans for using neural network diversification primarily consists in using above described diversification techniques in more sophisticated models in order to explore the possibility to improve training speed and reduce the number of training parameters. Popular architectures, such as transformers can benefit from the individual head diversification in multi-head attention block, as far as multiple heads are intended to learn various representation. Furthermore, we are planning to explore more pattern-oriented techniques for defining diversity between neurons to enable efficient diversification application in CNNs.

Acknowledgment

This research is supported by the Czech Ministry of Education, Youth and Sports from the Czech Operational Programme Research, Development, and Education, under grant agreement No. CZ.02.1.01/0.0/0.0/15003/0000421 and the Czech Science Foundation (GAČR 18-18080S).

References

- [1] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [2] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-second international joint conference on artificial intelligence*, 2011.
- [3] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.
- [4] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [5] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [6] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width, 2017.
- [7] Lodovico Marziale, Golden G. Richard, and Vassil Roussev. Massive threading: Using gpus to increase the performance of digital forensics tools. *Digital Investigation*, 4:73 – 81, 2007.
- [8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Nee-lakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
- [9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.
- [10] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.
- [11] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.
- [12] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *ICML*, 2011.
- [13] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.
- [14] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
- [15] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.
- [16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.
- [17] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [20] Lu Lu. Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, Jun 2020.
- [21] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

- [22] Hongyang Gao, Lei Cai, and Shuiwang Ji. Adaptive convolutional relus. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3914–3921, 2020.
- [23] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
- [24] Ben Luitjen, Regev Cohen, Frederik J de Bruijn, Harold AW Schmeitz, Massimo Mischi, Yonina C Eldar, and Ruud JG van Sloun. Deep learning for fast adaptive beamforming. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1333–1337. IEEE, 2019.
- [25] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.
- [26] Debesh Jha, Anis Yazidi, Michael A. Riegler, Dag Johansen, Håvard D. Johansen, and Pål Halvorsen. Lightlayers: Parameter efficient dense and convolutional layers for image classification, 2021.
- [27] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.
- [28] Gavin Brown. Diversity in neural network ensembles. Technical report, 2004.
- [29] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.
- [30] Naonori Ueda and Ryohei Nakano. Generalization error of ensemble estimators. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 90–95. IEEE, 1996.
- [31] Yijun Bian and Huanhuan Chen. When does diversity help generalization in classification ensembles?, 2021.
- [32] Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization, 2019.
- [33] Anders Krogh and Jesper Vedelsby. Validation, and active learning. *Advances in neural information processing systems 7*, 7:231, 1995.
- [34] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [35] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
- [36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [38] Akarachai Atakulreka and Daricha Sutivong. Avoiding local minima in feedforward neural networks by simultaneous learning. In Mehmet A. Orgun and John Thornton, editors, *AI 2007: Advances in Artificial Intelligence*, pages 100–109, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [39] Grzegorz Swirszczyk, Wojciech Marian Czarnecki, and Razvan Pascanu. Local minima in training of neural networks, 2017.
- [40] Yossi Arjevani and Michael Field. Symmetry and critical points for a model shallow neural network, 2020.
- [41] Kshitij Tayal, Chieh-Hsin Lai, Vipin Kumar, and Ju Sun. Inverse problems, deep learning, and symmetry breaking, 2020.