Take a Ramble into Solution Spaces for Classification Problems in Neural Networks

Enzo Tartaglione(✉) and Marco Grangetto

University of Torino, Turin, Italy
enzo.tartaglione@unito.it

Abstract. Solving a classification problem for a neural network means looking for a particular configuration of the internal parameters. This is commonly achieved by minimizing non-convex object functions. Hence, the same classification problem is likely to have several, different, equally valid solutions, depending on a number of factors like the initialization and the adopted optimizer.

In this work, we propose an algorithm which looks for a zero-error path joining two solutions to the same classification problem. We witness that finding such a path is typically not a trivial problem; however, our heuristics is able to succeed in such a task. This is a step forward to explain why simple training heuristics (like SGD) are able to train complex neural networks: we speculate they focus on particular solutions, which belong to a connected solution sub-space. We work in two different scenarios: a synthetic, unbiased and totally-uncorrelated (hard) training problem, and MNIST. We empirically show that the algorithmically-accessible solutions space is connected, and we have hints suggesting it is a convex sub-space.

Keywords: Neural networks · Solution space · Image classification

1 Introduction

One of the core problems in computer vision is image classification. Solving an image classification problem means being able to correctly recognize an image as being part of a class, which translates into the correct identification of key features. Image classification finds a number of direct applications, not restricted to tumor classification and detection [1], bio-metric identification [15,20,23], object classification [9] and even emotions [7]. This problem is typically complex to be solved, and a number of algorithms have been designed to tackle it [8,17,24]. However, the top-performance model is here represented by neural networks. In particular, the so-called convolutional neural networks (CNNs) are able to automatically take as input images, process them in order to extract the key features for the particular classification problem, and perform the classification itself. Applying very simple optimizing heuristics to minimize the loss

E. Ricci et al. (Eds.): ICIAP 2019, LNCS 11751, pp. 345–355, 2019.
https://doi.org/10.1007/978-3-030-30642-7_31

function, like SGD [4,27], slightly more complex optimizers like Nesterov [19] or Adam [14] moving to the sophisticated local entropy minimizer [2,6], it is nowadays possible to succeed in training extremely complex systems (deep networks) on huge datasets. Theoretically speaking, this is the "miracle" of deep learning, as the dimensionality of the problem is huge (indeed, these problems are typically over-parametrized, and the dimensionality can be efficiently reduced [25]). Furthermore, minimizing non-convex objective functions is typically supposed to make the trained architecture stuck into local minima. However, the empirical evidence shows that something else is happening under the hood: understanding it, in order to provide some warranty for all the possible applications of image classification, is critical.

In this work, we propose an heuristic approach which should help us to understand some basic properties of the found solutions in neural network models. Here, we aim to find a path joining two (or, in general, more) different solutions to the same classification problem. Early attempts to explore possible joining paths were performed using random walk-based techniques, but the complexity of the task, due to the typical high-dimensionality of the problem, made it extremely inefficient [13].

A recent work [12] suggests that solutions to the same problem are typically divided by a loss barrier, but a later work by Draxler et al. [10] shows the existence of low-loss joining paths between similar-performance solutions. Such a work, however, focuses on the loss function, which is a necessary but not sufficient condition to guarantee the performance on the training/test set. Our heuristics puts a hard constraint on it: we will never have a *performance* (evaluated as the number of samples correctly classified by the neural network model) below a fixed threshold. In the case we ask our model to correctly classify the whole training set, we will say we lie in the *solution region* S of the training model, also known as *version space*. This will be our focus along this work.

In the last few years, thanks to the ever-increasing computational capability of computers, bigger and bigger neural networks have been proposed, in order to solve always more complex problems. However, explaining why they succeed in solving complex classification tasks is nowadays a hot research topic [11,21,22]. Still, it is object of study why, using simple optimizers like SGD to minimize problems which are typically non-convex, is a sufficient condition to succeed in training deep models [5,16,18]. The aim of this work is to move a step in the direction of explaining such a phenomenon, analyzing some typical solutions to learning problems, and inspecting some properties of them. In this way, we aim to give some hints on which type of solutions SGD finds, guessing whether there is some room for improvement or not.

The rest of this paper is organized as follows. In Sect. 2 we set-up the problem environment, aim and the algorithm is illustrated and justified. Next, in Sect. 3 we test our algorithm on MNIST and on training sets containing uncorrelated, randomly-generated patterns. The experiments show that our proposed method is able to always find joining paths in S between any found solution for the same problem. Furthermore, hints on some properties of S are deducted studying

the joining path. Finally, Sect. 4 draws the conclusions and suggests further directions for future research.

2 The Proposed Algorithm

2.1 Preliminaries

In our setting, we have a *training set* Ξ_{tr} made of M pairs (ξ_i, σ_i), in which we identify the set of inputs ξ_i and their associated desired output σ_i. For the purpose of our work, we ask for some configuration W of the neural network such that the entire Ξ_{tr} problem is correctly solved. If such a condition is met, we say that the configuration W is a *solution* for the learning problem Ξ_{tr}. In other words, a weights configuration W_k is a solution when

$$y_i \,|\xi_i, W_k = \sigma_i \;\forall i \in \Xi_{tr} \tag{1}$$

If we define S as the subset of all the W configurations which solve the whole training problem Ξ_{tr}, we can say that $W_k \in S$. Let us imagine two solutions to the same problem Ξ_{tr}, W_a and W_b, are provided. We aim to find a path $\Omega_{ab} \subset S$ which joins W_a to W_b. At this point, we might face two different scenarios:

1. Ω_{ab} is simply a straight line. According to the work by Goodfellow et al., we could draw a straight line between W_a and W_b which might be parametrized, for example, as
$$l_{ab}(t) = (W_b - W_a)t + W_a \tag{2}$$
with $t \in [0, 1]$. According to this scenario, this is a sufficient condition to join the two different solutions. However, as showed by the same work of Goodfellow et al., this is not typical [12].
2. Ω_{ab} is a "non-trivial" path as $l_{ab} \not\subset S$, and is not a-priori guaranteed to exist. This is the typical scenario, and the setting in which we are going to work. The work by Draxler et al. [10] shows that there exists a path Γ_{ab} having low loss value, however, in general, $\Gamma_{ab} \not\subset S$. Our heuristics not only works for the case $\Omega_{ab} \neq l_{ab}$, but it guarantees $\Omega_{ab} \subset S$ (Fig. 1).

2.2 Finding the Path

Our heuristics generates the path Ω_{ab} in a "Markov chain" fashion: we are going to use a "survey" network W_x, whose task is to modify its configuration (i.e. the value of its parameters) in order to move from W_a's configuration to W_b, never leaving S. Hence, at time $t = 0$ we initialize $W_x = W_a$, and we ask W_x to explore S such that, at some time t_f, $W_x = W_b$. The exploration algorithm is designed according to three, very simple, basic concepts:

- We will never leave S.
- As we start from W_a, we want to arrive to W_b using a survey network W_x, which draws Ω_{ab} in t_f steps.

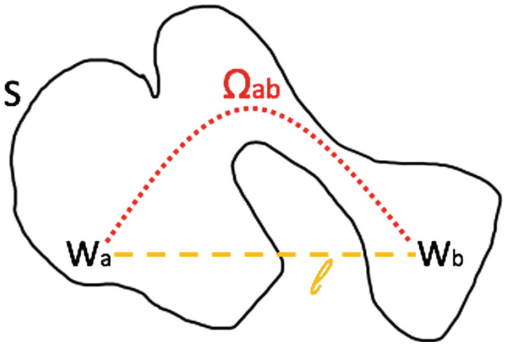


Fig. 1. Example of the "non trivial" scenario. Here, while l_{ab} goes out the solution region S, Ω_{ab} still remains inside it.

– At time t', W_x will just have knowledge of the training set, direction and distance towards the target W_b.

In order to reach W_b, we need to drive W_x at any time t to it. Towards this end, we use an elastic force:

$$\Delta W_x^t = \gamma(W_b - W_x^{t-1}) \tag{3}$$

where γ is an *elastic constant*, whose value is typically $\gamma \ll 1$.

If we just apply Eq. 3, in the non-trivial scenario, W_x will leave the solution region, as we will have $\Omega_{ab} \equiv l_{ab}$. Hence, what we need here is to change the trajectory for our Ω_{ab} in a "smart" way. It will be nice to stay away from the *frontier* of S. A local information we have, which might come handy in this context, is the gradient on the training set. If we perform a GD step, W_x should be naturally driven down the loss function and, supposedly, drives W_x away from the frontier of S.

Along with the elastic coupling and the GD step, we impose a norm constraint for W_x, acting as a regularizer, to be applied layer-by-layer, which bounds W_x's norm to:

$$n\left(W_x^l\right) = \|W_b^l\|_F - \frac{\|W_b^l\|_F - \|W_a^l\|_F}{\|W_b^l - W_a^l\|_F}\|W_b^l - W_x^l\|_F \tag{4}$$

where W_x^l indicates the l-th layer of W_x and $\|\cdot\|_F$ is the Frobenius norm. Essentially, we are imposing a linear constraint to the norm of W_x, which is function of the distance from W_b. Finally, as we have the hard constraint on remaining into S, we need to impose small steps for W_x

$$W_x^t = W_x^{t-1} + \delta W_x^t \tag{5}$$

where, typically,

$$\delta W_x^t \ll \nabla W_x^{t-1} \tag{6}$$

In this way, unless we find a local minimum very close to W_b and exactly on the same path followed by W_x (extremely unlikely as empirically observed, issue

which can be anyway easily tackled with a proper tuning of γ), we avoid to get stuck in any local minimum.

To sum-up, in order to generate Ω_{ab}, after we have initialized W_x to W_a, we iteratively perform the following steps:

1. Apply an elastic coupling in the direction of W_b (Eq. 3), with the hard constraint of never leaving S (this is hard because we will simulate a "hitting a wall"-like fashion, i.e. we will discard all the steps which will put W_x outside S)
2. Perform N_{epochs} of gradient descent (GD) steps evaluated on Ξ_{tr}
3. Properly normalize W_x (Eq. 4)

The general algorithm is summarized in Algorithm 1.

Algorithm 1. Find joining path between W_a and W_b

procedure TRACK $\Omega(W_a, W_b, \Xi_{tr})$ ▷ Implicitly, W_x always normalized as in Eq. 4
  $W_x = W_a$
  $\Omega = W_x$
  while $W_x \neq W_b$ **do**
    for N_{epochs} **do**
      $W_x = W_x - \eta \nabla W_x$ ▷ ∇W_x computed on Ξ_{tr}
      if $W_x \notin S$ **then**
        return $\emptyset$ ▷ η, γ not properly set
    W_x-tmp $= W_x - \gamma(W_b - W_x)$
    if W_x-tmp $\in S$ **then**
      $W_x = W_x$-tmp
    $\Omega =$ append(Ω, W_x)
  return Ω

2.3 Properties of the Path

Once we have obtained Ω_{ab}, we can perform an empirical investigation on it. There are some interesting observations we can perform on it:

– Is there any property related to the shape of S? As typical problems are extremely high-dimensional, it is very difficult to deduct some global property on S. However, we might have some hint on how S is shaped from two indicators:
  - If we are always able to find $\Omega_{ab} \subset S$, then we might suggest that all the algorithmically-accessible solutions in S, collected in $S_{algo} \subset S$, live in a connected subspace.
  - We can study the Hessian along Ω_{ab}. Even though this is not a fully-informative observation for S, we can deduce some properties, like the shape of the loss in S_{algo}.

– Verify how the loss function varies along Ω_{ab}: as our technique is strictly bounded to S and not necessarily to the minimization of the loss function, it may happen that some solutions to Ξ_{tr} have high loss.
– Check how the generalization error, defined as the error on the test set, varies along Ω_{ab}.

All of these aspects will be empirically investigated in Sect. 3.

3 Experiments

The proposed algorithm was tested under two very different settings and architectures. In both cases, a $\Omega_{ab} \subset S$ path has always been found.

3.1 Tree Committee Machine on Random Patterns

In our first experiments, we use a simple kind of neural network, the so-called Tree Committee Machine (TCM). It is a binary classifier, consisting in one-hidden neural network having N inputs and K neurons in the hidden layer. The connectivity of the hidden layer is here tree-like: each k-th neuron of the hidden layer is able to receive data from an exclusive $\frac{N}{K}$ subset of the input. In particular, for our setting, the general output of the TCM is defined as

$$\hat{y}_i = \tanh\left[\sum_{k=1}^{K} \text{htanh}\left(\sum_{j=1}^{\frac{N}{K}} W_{kj} \cdot \xi^{\mu}_{k\frac{N}{K}+j}\right)\right] \tag{7}$$

where htanh is the hard tanh.

The training set Ξ_{tr} is randomly generated: the input patterns $\xi_i \in \{-1; +1\}^{(N\times M)}$ and random desired outputs $\sigma \in \{-1, +1\}^{M}$.

All the experiments here shown are performed on TCMs having size $N = 300$ and $K = 3$ and the training sets consist in $M = 620$ samples. The training of the reference solutions to Ξ_{tr} has been performed using the standard GD technique, minimizing the binary cross-entropy loss function, with $\eta = 0.1$. The network has been initialized using a gaussian initializer. In this setting, $\gamma = 0.001$ and $N_{epochs} = 5$. The algorithm was tested on 10 different, randomly-generated datasets, and for each of them 3 different configurations $W_i \in S$ were obtained and attempted to be connected. The implementation of the neural network and of the algorithm is in Julia 0.5.2 [3].

Even though we are in the typical scenario for which the error on $l_{ab} > 0$, we are able to find a non-trivial path in S. For this network, it is also possible to compute the exact Hessian matrix. Surprisingly, the typical observed scenario here is that, along any found $\Omega \subset S$, all the non-zero eigenvalues of the Hessian matrix are strictly positive, the cardinality of non-zero eigenvalues is constant and the reference solutions represent local minima for the trace of the Hessian matrix. An example of this observed result is shown in Fig. 2. This result is obtained in a hard learning scenario, and may suggest us that, even though the learning problems are typically non-convex, GD-based techniques work because the algorithmically accessible S_{algo} region, is convex.

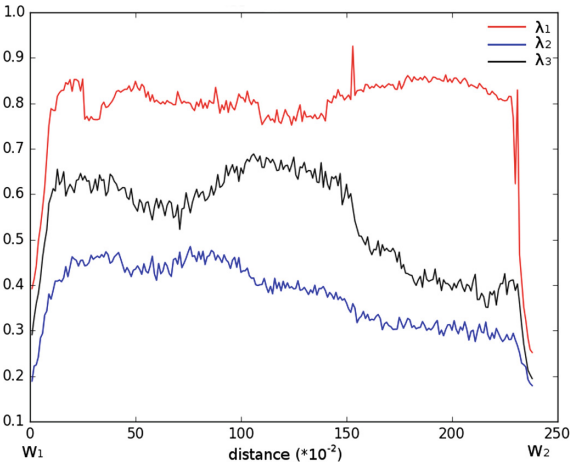
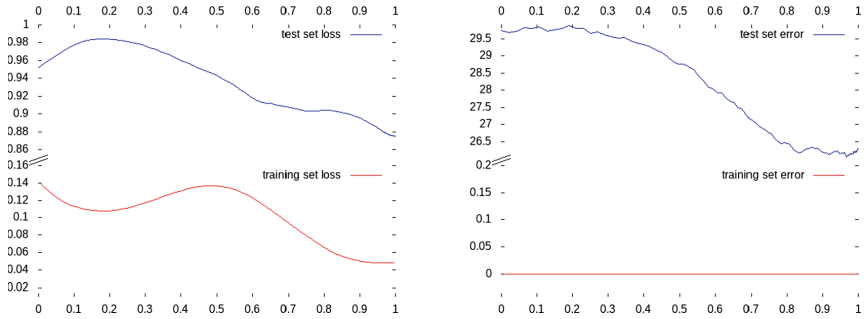


Fig. 2. Hessian eigenvalues along Ω_{ab} in TCM for random patterns. In this case, just three eigenvalues are non-zero and all positive. Along the path, the loss on the training set is proportional to the trace of the Hessian.

3.2 LeNet5 on MNIST

Experiments on LeNet5 solutions trained on the MNIST dataset have been performed. In particular, at first simulations on a reduced MNIST are shown (training is performed on the first 100 images: we are going to call it MNIST-100) and on the full MNIST dataset. The software used for the following simulations is PyTorch 1.1 with CUDA 10.



(a) Loss on training and test set in Ω_{ab} (b) Error [%] on training and test set in Ω_{ab}

Fig. 3. Example of Ω_{ab} for LeNet5 with MNIST-100. The x axis is a normalized distance between W_a and W_b.

For the MNIST-100 case, the networks have been trained using SGD with $\eta = 0.1$, and initialized with Xavier. The joining path heuristic used $\gamma = 0.001$ and

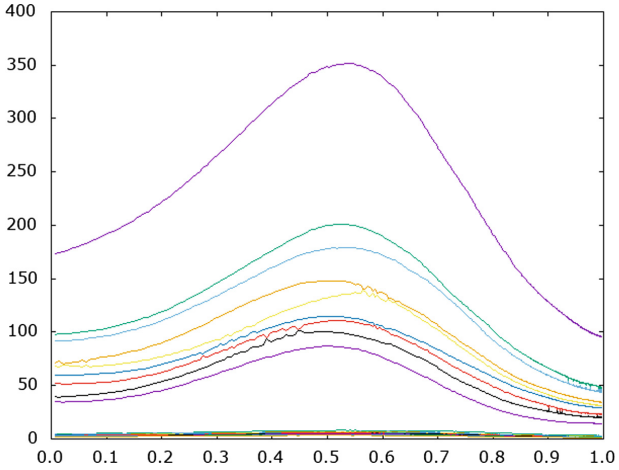
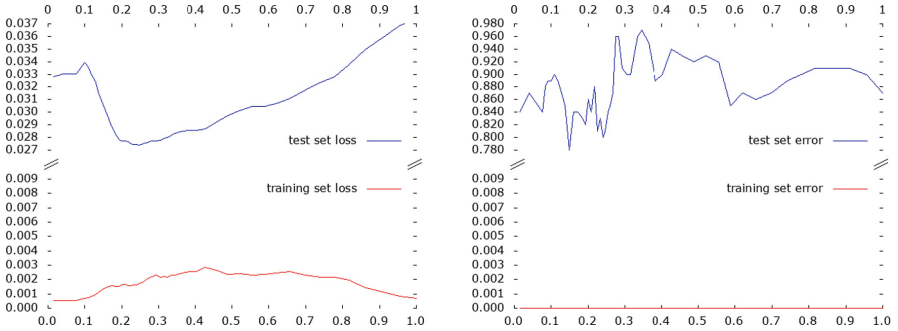


Fig. 4. Hessian eigenvalues along Ω_{ab} for LeNet5 trained on MNIST-100 (same experiment as Fig. 3). Here the top-20 Hessian eigenvalues are plotted.



(a) Loss on training and test set along Ω_{ab} (b) Error [%] on the test set along Ω_{ab}

Fig. 5. Example of Ω_{ab} for LeNet5 with the entire training set. The x axis is a normalized distance between W_a and W_b.

$N_{epochs} = 5$. Despite the higher dimensionality and complexity of LeNet5, also in this case it has always been possible to find a Ω_{ab} path in S. A typical observed behavior is shown in Fig. 3. It is here interesting to observe that in general, moving through Ω, both the training and test loss are no longer monotonic or bi-tonic, but they show a more complex behavior (an example is in Fig. 3(a)). Furthermore, observing the test set error, it shows a similar behavior to the test set loss, but not locally exactly the same (Fig. 3(b)).

It can be here interesting to investigate the behavior of the eigenvalues of the Hessian along Ω_{ab} also in this scenario. Figure 4 is a plot for the top-20 eigenvalues. The Hessian eigenvalue computation has been performed here using

the code by Gholami [26]. Interestingly, even for a more complex architecture like LeNet5, along the entire Ω_{ab} path, the top eigenvalues are all positive.

Besides the simulations on MNIST-100, we also attempted to find a joining path between two solutions for the entire MNIST dataset, still using LeNet5. In this case, $\gamma = 0.1$ and $N_{epochs} = 5$, while the training of the initial configuration is performed using SGD with $\eta = 0.1$ and initializing with Xavier. According to our findings, in this setting, a zero-error joining path, even for the whole MNIST problem, typically exists (Fig. 5). Interestingly, the best generalization performance (at about 0.2 in the normalized distance scale) is here found far from both the solutions found by SGD, and typically can not be found by vanilla-SGD, as there is a higher training loss value (even if it lies in the version space).

4 Conclusion

In this work, a heuristic approach to find a path Ω_{ab} joining two solutions W_a and W_b to the same training problem Ξ_{tr} is proposed. The main property of Ω_{ab} is that it entirely lies in the solution space S of the W's configurations which solve the training problem. In general, such an approach is not guaranteed to produce an Ω: if S is not connected and W_a and W_b belong to two different sub-spaces of S, by construction, Ω_{ab} can not exist. By our empirical observations, with a randomly-generated, uncorrelated, synthetic training set and with MNIST, the subspace $S_{algo} \subseteq S$ accessed by GD-based techniques seems to be connected. Furthermore, we have some hints indicating that S_{algo} might be convex and a further proof that SGD alone is not sufficient to guarantee the best generalization, neither for nowadays simple classification problems like MNIST.

The proposed technique potentially allows us to extend the investigation of S also to non-typical algorithmic solutions to the learning problem, along the drawn Ω paths. These findings open to new researches in the field of explainable neural networks. Future work involves the study of how the generalization error varies along Ω on more complex classification tasks and the design of an algorithm to boost it.

References

1. Al-Shaikhli, S.D.S., Yang, M.Y., Rosenhahn, B.: Brain tumor classification using sparse coding and dictionary learning. In: 2014 IEEE International Conference on Image Processing (ICIP), pp. 2774–2778. IEEE (2014)
2. Baldassi, C., Ingrosso, A., Lucibello, C., Saglietti, L., Zecchina, R.: Local entropy as a measure for sampling solutions in constraint satisfaction problems. J. Stat. Mech: Theory Exp. **2016**(2), 023301 (2016)
3. Bezanson, J., Karpinski, S., Shah, V.B., Edelman, A.: Julia: a fast dynamic language for technical computing. arXiv preprint arXiv:1209.5145 (2012)
4. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Lechevallier, Y., Saporta, G. (eds.) Proceedings of COMPSTAT 2010, pp. 177–186. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-7908-2604-3_16

5. Brutzkus, A., Globerson, A., Malach, E., Shalev-Shwartz, S.: SGD learns over-parameterized networks that provably generalize on linearly separable data. arXiv preprint arXiv:1710.10174 (2017)
6. Chaudhari, P., et al.: Entropy-SGD: biasing gradient descent into wide valleys. arXiv preprint arXiv:1611.01838 (2016)
7. Chen, M., Zhang, L., Allebach, J.P.: Learning deep features for image emotion classification. In: 2015 IEEE International Conference on Image Processing (ICIP), pp. 4491–4495. IEEE (2015)
8. Chen, Y., Nasrabadi, N.M., Tran, T.D.: Hyperspectral image classification via kernel sparse representation. IEEE Trans. Geosci. Remote Sens. **51**(1), 217–231 (2013)
9. Doulamis, N., Doulamis, A.: Semi-supervised deep learning for object tracking and classification. In: 2014 IEEE International Conference on Image Processing (ICIP), pp. 848–852. IEEE (2014)
10. Draxler, F., Veschgini, K., Salmhofer, M., Hamprecht, F.A.: Essentially no barriers in neural network energy landscape. arXiv preprint arXiv:1803.00885 (2018)
11. Frosst, N., Hinton, G.: Distilling a neural network into a soft decision tree. arXiv preprint arXiv:1711.09784 (2017)
12. Goodfellow, I.J., Vinyals, O., Saxe, A.M.: Qualitatively characterizing neural network optimization problems. arXiv preprint arXiv:1412.6544 (2014)
13. Ishihara, A.K., Ben-Menahem, S.: Control on landscapes with local minima and flat regions: a simulated annealing and gain scheduling approach. In: 47th IEEE Conference on Decision and Control, CDC 2008, pp. 105–110. IEEE (2008)
14. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
15. Klimanee, C., Nguyen, D.T.: Classification of fingerprints using singular points and their principal axes. In: 2004 International Conference on Image Processing, ICIP 2004, vol. 2, pp. 849–852. IEEE (2004)
16. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: Advances in Neural Information Processing Systems, pp. 6389–6399 (2018)
17. Li, J., Najmi, A., Gray, R.M.: Image classification by a two-dimensional hidden markov model. IEEE Trans. Signal Process. **48**(2), 517–533 (2000)
18. Li, Y., Liang, Y.: Learning overparameterized neural networks via stochastic gradient descent on structured data. In: Advances in Neural Information Processing Systems, pp. 8157–8166 (2018)
19. Nesterov, Y., Polyak, B.T.: Cubic regularization of newton method and its global performance. Math. Program. **108**(1), 177–205 (2006)
20. Rattani, A., Derakhshani, R., Saripalle, S.K., Gottemukkula, V.: ICIP 2016 competition on mobile ocular biometric recognition. In: 2016 IEEE International Conference on Image Processing (ICIP), pp. 320–324. IEEE (2016)
21. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Advances in Neural Information Processing Systems, pp. 3856–3866 (2017)
22. Samek, W., Wiegand, T., Müller, K.R.: Explainable artificial intelligence: understanding, visualizing and interpreting deep learning models. arXiv preprint arXiv:1708.08296 (2017)
23. Soleymani, S., Torfi, A., Dawson, J., Nasrabadi, N.M.: Generalized bilinear deep convolutional neural networks for multimodal biometric identification. In: 2018 25th IEEE International Conference on Image Processing (ICIP), pp. 763–767. IEEE (2018)

24. Tarabalka, Y., Benediktsson, J.A., Chanussot, J.: Spectral-spatial classification of hyperspectral imagery based on partitional clustering techniques. IEEE Trans. Geosci. Remote Sens. **47**(8), 2973–2987 (2009)
25. Tartaglione, E., Lepsøy, S., Fiandrotti, A., Francini, G.: Learning sparse neural networks via sensitivity-driven regularization. In: Advances in Neural Information Processing Systems, pp. 3878–3888 (2018)
26. Yao, Z., Gholami, A., Lei, Q., Keutzer, K., Mahoney, M.W.: Hessian-based analysis of large batch training and robustness to adversaries. In: Advances in Neural Information Processing Systems, pp. 4949–4959 (2018)
27. Zinkevich, M., Weimer, M., Li, L., Smola, A.J.: Parallelized stochastic gradient descent. In: Advances in Neural Information Processing Systems, pp. 2595–2603 (2010)