Important Milestones in the Study of Neural Networks with Random Weights

Jürgen Brauer
Faculty of Computer Science
University of Applied Sciences Kempten
Germany
juergen.brauer@hs-kempten.de

Abstract Neural networks with partially random weights are currently not really an independent field of research. However, the first works on random neural networks date back to the 1990s and in the last three decades there have been important new works in which random weights have been used and which are promising in that they give surprisingly good results when compared to approaches in which all weights are trained. These works, however, come from very different subareas of neural networks: Random Feedforward Neural Networks, Random Recurrent Neural Networks and Random ConvNets. In this paper, I analyze the most important works from these three areas and thereby follow a chronological order. I also work out the core result of each work. As a result, the reader can get a quick overview of this field of research.

Keywords: Random Neural Networks, Random Convolutional Neural Networks, Deep Learning, Neural Networks

1 Introduction

As soon as one uses random numbers in an algorithm, we speak of a randomized algorithm or a Monte-Carlo approach. Many successful algorithms in computer science are such Monte-Carlo approaches and show the great benefits of using randomness. For example, the Particle Swarm Optimizer (PSO) uses a large amount of particles, which in each update step move a little bit in the direction of the previously found optimum and at the same time also a little bit in a random direction, in order to scan the search space on the way to the supposed optimum. Another example for a successful Monte-Carlo algorithm is the particle filter. Probable system states in a system state estimation problem are represented here by a set of weighted particles. At the beginning, the particles are typically distributed randomly in the state space and then iteratively pass through two essential steps: a so-called measurement correction step, where the particles are redistributed in the state space due to new measured values and a so-called prediction step, where knowledge about the system is used to shift the particles also in the direction of more probable states. In an importance resampling step random numbers are used to calculate which particles should be retained in the population in the next step. Here, randomness helps to maintain the variance of possible solutions.

Neuronal networks are no exception to this rule. It is also known that randomness can help here. Neu-

ronal networks are typically initialized randomly and it has been shown that some random distributions can be more helpful than others for the initialization of weights. Dropout within a Multi Layer Perceptron, for example, is a technique that randomly decides for each individual neuron in each training step whether the neuron is included or ignored in that training step. In contrast to this, the technique of data augmentation uses randomness outside the model: the actual training data is augmented by random transformations in order to increase the variance of the training data and thus make the model itself invariant to transformations of the input data in the inference phase.

At the beginning of the 1990s, a first work in the field of neural networks was presented [21], which showed that layers with randomly initialized weights, which are not trained in the following, can have advantages. In the following three decades there were several other works in the field of neural networks with random weights - from different areas and with very different architectures [11], [29], [1], [5], [31], [13], [19], [4], [14], [28], [7], [20], [15], [24], [33], [22]. However, random neural networks have not really established themselves as an independent field of research. It should be noted that random neural networks almost never use completely random weights: usually, some layers of random weights are used in combination with layers where the weights are trained (e.g., "readout" layers).

In this paper I would like to offer the interested reader a compact overview of the field of random neural networks. I present the most important works from the field of random feedforward neural networks, random recurrent neural networks and random convnets. Besides the presentation of the respective approaches of these works, I also describe the main results of the works in a compact way. This is intended to give the reader a comprehensive and targeted overview of the work on random neural networks.

There are several previous survey papers on random NN. The review by Scardapane and Wang [25] (2017) and the review by Cao et al. [3] (2018) both present a broad overview. In contrast, a recent review by Gallicchio and Scardapane [8] (2020) focuses especially on attempts at extending the ideas of random NN to deep neural architectures with many layers.

All these previous surveys, however, represent broad representations of these topics. In contrast, I only select the most important milestones in this area here. Further, I present the respective work in more detail and highlight the most important results of each work.

2 Random Feedforward Neural Networks

1992: Random Vector Functional Link Network (RVFL). The review by Cao et al. [3] shows that NN with random weights are not a new idea. Pao and Takefuji [21] explored the idea of using random weights in a feed-forward network already in 1992. They introduced the idea of *Random Vector Functional Link Networks (RVFL)*. A RVFL is a simple feedforward network with a single hidden layer and an output layer, but in contrast to a Multi-Layer-Perceptron (MLP), the output layer does not only get its input from the previous hidden layer, but also from the input layer. So there are what is nowadays called a *skip-connections*. Further, in a RVFL the weights from the input layer to the hidden layer are not trained, but only the weights from the input and hidden layer to the output layer. They underline the success of this random NN model by showing that RVFLs have been used in many real-world applications, e.g., time-series data prediction and English language handwritten script recognition. In 1995, Igelnik and Pao [12] could prove that the RVFL model is a universal

function approximator as the MLP is, provided the function to be approximated satisfies certain smoothness conditions. **Contribution:** This work showed a new way for research: neural networks with partly random weights make sense and have some advantages.

2004: Extreme Learning Machine (ELM). Inspired by the RVFL model, Huang et al. [11] explored in a series of works feed-forward NN with partially or fully connected multiple layers. In contrast to the RVFL model no skip connections are used in ELMs. In ELMs all the weights to hidden neurons are randomly set and only the weights to the output layer are computed using the Moore-Penrose inverse. This makes learning extremely fast and hence the name *Extreme Learning Machine*. In 2006, Huang et al. [10] presented a work which proved the universal function approximation capability of ELMs with a single hidden layer for almost any nonlinear piece-wise continuous function that is used as activation function for the hidden nodes. **Contribution:** Here it was shown that random weights to hidden layers in the the standard multi layer perceptron (MLP) architecture and random weights in feedforward neural networks with partially connected layers also work. It was also shown that an iterative learning approach like gradient descent is not absolutely necessary, if the weights to the output neurons can be calculated.

2013: The No-Prop algorithm. Very similar to ELM, Widrow et al. [29] proposed an approach in 2013 which they called *No-Prop*. As for ELM, the No-Prop approach also uses several hidden layers, where the weights to the hidden neurons are assigned randomly and are fixed. Only the output weights are trained. But while ELM uses the Moore-Penrose inverse in a batch-fashion in order to compute the weights to the output units, the No-Prop approach uses the iterative LMS (Least Mean Square) algorithm, which is also called *delta rule* or *Widrow-Hoff rule* - a weight update rule for a neural network with only a single layer. While the Perceptron learning rule was designed for neurons with a Heaviside step activation function, the delta rule can be considered as a more general solution, since it can be applied to neurons with general activation functions. So the central difference between ELMs and the No-Prop approach is the way how the output weights are trained, while another difference is that ELMs by definition also can consist of partially connected layers.

The authors also compared No-Prop with Back-propagation (Back-Prop) trained networks for the classification of 200 different Chinese characters. 50 noisy versions of the 200 training patterns were used, resulting in 10.000 training patterns. Models were trained with Back-Prop and with No-Prop and then tested on 100 test characters. The results showed, that when the number of training patterns is less than the LMS network capacity, the No-Prop network and the Back-Prop network performed equivalently. Here, the LMS network capacity of a network was defined as the maximum number of distinct patterns that can be trained into a No-Prop network with zero error. When training over capacity, Back-Prop performed better than No-Prop. But by increasing the number of hidden neurons, i.e., by increasing the LMS network capacity, the performance of No-Prop was similar to that of the Back-Prop algorithm. **Contribution:** An important contribution of this work is the interesting comparison between a random neural network where the output layer is trained with No-Prop and a non-random neural network trained with the standard Back-Prop method.

2016: Comparison of ELM with No-Prop. Alshamiri et al. [1] compared the classification accuracy of ELM networks with randomized feed-

forward networks trained using No-Prop. For their evaluation five standard benchmarks for classification were used and a range of different number of neurons for the hidden randomized layer was considered ([10, 20,..., 190, 200] neurons). The results showed that the No-Prop algorithm provides good generalization performance and that this performance is largely independent from the number of hidden neurons. The authors claimed that No-Prop was rather slow compared to ELM, but unfortunately there are no numbers regarding computation time given in their paper. ELM showed a poor generalization performance on the benchmarks. However, a third approach was also considered, which is a minor variant of the ELM approach with a regularization term. This regularized ELM approach achieved approximately the same test accuracy values as the No-Prop model and even outperformed No-Prop on some of the benchmarks. **Contribution:** ELM random networks seem to generalize much worse compared to No-Prop trained random networks.

2019: Weight Agnostic Neural Networks. Gaier and Ha [5] analyzed the performance of network architectures that were trained in an evolutionary fashion. Starting with very simple networks in the first generation 1.) new nodes are inserted into the network, 2.) connections between nodes are added, or 3.) activation functions of nodes are changed. For 3.) the authors considered a large range of possible activation functions including linear, step, sin, cosine, Gaussian, tanh, sigmoid, absolute value, invert and ReLU. Since the focus of their work was to search for good architectures and not for good weights, the weights were not trained at all, i.e., random weights were used. Among others two different conditions were considered in the experiments: 1.) experiments, where the weights to the nodes were initialized randomly and 2.) experiments, where there was only a single random value which was assigned to all weights. The experimental evaluation was carried out using three continuous control tasks: 1.) given a cart-pole system, a pole must be swung from resting to upright position and then balanced, 2.) a two-legged agent has to learn to walk through randomly generated terrain, and 3.) a two-dimensional car-racing environment. The results were surprising in the sense that even evolved networks with *a single shared random weight could solve such complex control tasks*. By evaluating different weight values from the interval [-2,2] the authors also showed that the evolved architectures were not sensitive to the value of this single shared weight. **Contribution:** This work made a huge contribution by rendering the importance of weights of neural networks in a new light since it suggests that it is more the topology and the function of the nodes which are important for the performance of such networks and not the specific weight values.

2019: Random topologies, but data transformation is learned. Xie et al. [31] explored randomly wired neural networks for image recognition. The interconnection topology of these random neural networks is automatically generated. Three different stochastic graph generators are used to produce graphs with different topologies. E.g., the Watts-Strogatz (WS) graph generator is used to generate graphs with small-world properties. A generated graph is then transformed into a directed acyclic graph (DAG). Edges in this directed graph define how the data (tensors) flows from one node to another. Each node in the graph transforms an input tensor to an output tensor, where the data transformation is learned. The nodes can have a different number of input and output edges and each node performs three steps: an aggregation, a transformation and a distribution of the data. In the aggregation step the input data is first combined via a weighted sum where the weights are learn-able. In

the transformation step the data is then processed by a ReLU-convolution-BatchNormalization triplet, where the same convolutional filter size is used for all nodes. In the final distribution step the node sends out a copy of the transformed data over all the output edges to subsequent nodes in the graph. *Surprisingly, several variants of these randomly generated neural networks had competitive and even better accuracy on the ImageNet benchmark (classification task) when compared with hand-designed ConvNets*. E.g., when considering only networks with less than 600 MFLOPs, a randomly generated neural network achieved 74.7% top-1 accuracy compared to 74.7% by MobileNet v2 [23] and 74.9% by Shuffle-Net v2 [18]. A randomly generated network with a number of FLOPs similar to that of ResNeXt-101 [30] even outperformed this model: 80.1% (random NN) compared to 79.5% (ResNeXt-101) top-1 accuracy. **Contribution:** This work showed that neural networks with randomly generated topologies (note that the weights are learned here) can perform as well as hand-designed network topologies.

3 Random Recurrent Neural Networks

2001/2002: Reservoir Computing is born. In 2001 Jaeger introduced the *Echo State Network (ESN)* [13]. A similar idea was introduced as the *Liquid State Machine (LSM)* by Maass et al. [19] in 2002. Both approaches share a common idea: an input signal is mapped with the help of a recurrent neural network with a random topology and random weights to a sequence of new states. This random recurrent neural network is called a *reservoir*, since it provides a reservoir of neuronal firing dynamics which has shown to be helpful as a (pre-)processing step for many tasks as, e.g., classification tasks. Another distinct set of neurons are used as output neurons. The tasks of these output neurons is to read the state of the reservoir and map it to a desired output state. Note, that only the weights from the reservoir neurons to these output neurons are trained (e.g., by linear regression), while the connections between neurons within the reservoir are randomly initialized and fixed. The principle of reservoir computing, to perform a non-linear mapping of input data before further processing, is very similar to the approach of kernel-based learning methods like Support Vector Machines (SVM) [2], where input vectors are mapped with the help of a kernel function to a higher-dimensional feature space where it is much easier to find a separating hyperplane (for classification problems). While both the ESN and the LSM share the central idea of exploiting the non-linear dynamics of a reservoir, the ESN can be discriminated from the LSM approach by the property that the LSM approach focuses on biological modeling and uses spiking neurons. Reservoir computing can also be considered as an approach to circumvent difficulties when training recurrent neural networks: Instead of training the weights, the weights are fixed and used with their randomly initialized values.

The reservoir has to have some properties in order to provide a helpful mapping from the input to a feature space, which can then be used by the readout neurons. One important property is that the neurons have to provide a nonlinear mapping, e.g., by using fire-rate model neurons with a non-linear activation function. Another property is related to the wish that the reservoirs shall store information within the recurrent loops in the network such that it has an influence onto the computations in the next processing step. In order to achieve this, the network has to have the *Echo State Property (ESP)*. A network with this property asymptotically eliminates any information from the initial conditions. In other words: The reservoir has to have a ”fading memory”. Different variants of reservoirs have been

analyzed in literature, e.g., ESNs with additional direct trainable input-to-output connections, or, e.g., ESNs with output neurons to reservoir neurons feedback connections. **Contribution:** Both the ESN and the LSM model have shown: the dynamic firing patterns of random recurrent neural networks can provide a beneficial representation for further processing.

2003: Reservoir computing with a physical reservoir. Reservoirs can be implemented by simulating a recurrent neural network in a computer. However, some attractiveness of the reservoir computing approach goes back to the possibility to use even physical reservoirs. The central idea here is to use a physical random nonlinear excitable medium which provides a high-dimensional dynamical response to an input signal ("echo"). Many natural systems can be used as physical reservoirs, since many of them show such a non-linear response when an input signal is given. In a pioneering work with the title "Pattern recognition in a bucket" from 2003 the benefit of such a physical reservoir for pattern classification was shown by Fernando and Sojakka [4]. In this work a simple discriminative classifier for the words "one" vs. "zero" was built with the help of a bucket of water. A glass bucket of water was placed on top of an over-head projector. The audio input signal of samples of the word "one" or "zero" was first pre-processed using a short-time Fourier transform in order to compute the underlying frequencies of the audio signal. The frequency spectrum was then used as input and translated into movements of 8 electrical motors which stimulated the water surface. Each of the 8 motors was driven by another part of the frequency spectrum. The water - as a natural system with non-linear dynamics - then responses to this input signal with waves that non-linearly interact with each other. The resulting interference patterns were filmed and used as input to a perceptron classifier. In a recent review from 2019 Tanaka et al. [27] give an overview of different approaches to design physical reservoirs. This work shows that a large number of very different reservoir systems have already been tried out: Authors have used electronic, photonic, spintronic, mechanical and even biological reservoirs in the context of reservoir computing. **Contribution:** The idea of using random dynamical systems to (pre-)process an input signal is not restricted to simulations with artificial neural networks. Physical reservoirs can be used as well.

2007: Stacked Echo State Networks with additional top-down information flow. Since many time series as texts or speech show a multi-scale characteristics, Jaeger [14] proposed a hierarchical architecture, called *Dynamical Feature Discoverer (DFD)*, where each layer consists of an ESN and is supposed to represent dynamical features on different temporal scales. This is the first experimental work on stacked reservoirs. This architecture provides two flows of information: A bottom-up flow of information, where increasingly coarser features are extracted from the input signal and a top-down flow of information, where feature expectations are passed down. Feature expectations are generated by each level in the architecture in the form of vote vectors. These vote vectors are used as weights to combine the next-lower-level features vectors into a final representation of the input signal at that level. While the feature vector can be considered as a finer-grained representation of the signal, the vote signal that comes from the layer above can be seen as some abstraction of the feature signal. **Contribution:** Hierarchical representations are probably one of the key elements for the success of Deep Learning. This first work from the field of reservoir computing showed that hierarchical reservoirs, i.e., hierarchies of random

recurrent neural networks, work as well.

2010: Stacked Echo State Networks with a readout layer for each reservoir. Triefenbach et al. [28] also experimented with a hierarchy of Echo State Networks for the task of phoneme recognition. But in contrast to the previous work by Jaeger [14], the authors used a readout layer after each of the reservoirs which builds the input for the next reservoir layer. Experiments on the TIMIT benchmark showed that such a layered hierarchy of reservoirs can yield better results when compared to a shallow (one-layer) reservoir. **Contribution:** This work showed for the first time that hierarchical random recurrent neural networks work better than shallow random recurrent neural networks.

2017: The vanilla version of stacked ESNs: Deep Echo State Networks (deep ESN). While Jaeger [14] and Triefenbach et al. [28] already explored the idea of stacked ESNs, Gallicchio and Micheli [7] experimented with the vanilla version of stacked ESNs. In their work they experimented with multiple reservoir layers which are stacked one on top of each other - without such an additional mechanism like vote vectors as in the DFD architecture [14] and in contrast to [28] only one readout layer was used on top of the highest reservoir layer. The first reservoir layer is fed by the input signal. Successive layers are then fed by the output of the previous one. The new model is called *deepESN (deep Echo State Network)*. The authors described also different versions of such a deep ESN: The *deepESN-IA (input-to-all)* projects the input not only to the first reservoir layer, but also to all other reservoir layers. For comparing such models of stacked reservoirs with shallow variants, the authors also considered *groupedESN*, where also multiple reservoirs are used and each reservoir receives the input, but the reservoirs are not stacked on top of each other in this variant. Experiments were done with a small dataset of two sequences in order to investigate the extent of time-scales differentiation among the different reservoir layers. The spectral radius of the weight matrix is the largest absolute value of its eigenvalues. Both, the spectral radius and the leak factor of the leaky-integrator neurons determine how long old inputs influence new states of the reservoir. In order to to separate the influence of these hyperparameters from architectural aspects (stacked vs. not-stacked, input projects to first reservoir only vs. input projects to all reservoirs), both the spectral radius and the leak factor was set to the same value for each reservoir layer. In the experiments the first input sequence for the models consisted of 5000 elements uniformly drawn from an alphabet of 10 elements, while the second sequence only differed from this first one by another value for step 100. The difference in the development of the resulting states of the reservoirs after this step 100 was then observed. The results showed that a stacked reservoir architectures show a much larger time-scale differentiation which suggests that a model with stacked reservoirs has the ability to show a larger diversification of temporal representations of sequences. Note that in 2017, also another work presented by Malik et al. [20] proposed the vanilla version of stacked ESNs under another term (*Multilayered Echo State Machine (ML-ESN)*) which was experimentally evaluated on seven different benchmarks and outperformed the shallow ESN approach with just one reservoir layer. In a recent survey, Gallicchio and Micheli [6] summarize the advances in the field of deep reservoir computing with a focus on the Deep ESN model and its variants. **Contribution:** Hierarchies of random recurrent neural networks can better represent a large quantity of different temporal sequences compared to shallow reservoirs.

4 Random ConvNets

2009: One and two layer CNNs with random filters. Jarett et al. [15] analyzed in a large set of experiments on the Caltech-101, NORB, and the MNIST dataset 1.) the influence of different non-linear activation functions, 2.) whether there is an advantage of a two-layer feature hierarchy compared to a single-layer feature hierarchy, and 3.) compared random conv layers with conv layers trained unsupervised or supervised. Surprisingly, their results show that a two-layer CNN with random filters yielded a high 62.9% accuracy on the the Caltech-101 dataset. The best accuracy was achieved by unsupervised pre-training of the filters followed by supervised refinement. However, the improvement over purely supervised training is rather small. The authors also computed the optimal input patches for the random filter bank in the first layer and found that these input patterns are very similar to the optimal inputs when using a learned filter bank in the first layer. Thereby they showed, *that a random filter bank can be frequency and orientation selective*. **Contribution:** This work showed: random filter banks work nearly as good as trained filter banks.

2011: One layer convolutional square-pooling architecture with random filters. Saxe et al. [24] already provided 2011 a mathematical analysis why filters with random weights perform so well. They proved mathematically that even filters with random weights are frequency selective and underlined that frequency selectivity is an important ingredient for object recognition systems. An experimental evaluation was presented on the CIFAR-10 and the NORB dataset where CNNs with random filters and CNNs with trained filters were compared. For the NORB dataset, they trained 110 models which used random filter weights. 110 other models where also trained but the filters were pre-trained using TICA, a feature-learning algorithm introduced by Le et al. [17], and then filters were fine-tuned for better discrimination using the normal Backprop algorithm. For the CIFAR-10 experiments 55 models with random filters and 55 models with filter tuning were trained. The results on the CIFAR-10 dataset showed that fine-tuning the filters only gave an accuracy advantage of about 6%. The results on the NORB dataset showed that some of the top random architectures could achieve even a slightly higher accuracy (89.6%) compared to the top fully trained architectures (86.5%). While this work of Saxe et al. [24] is highly interesting, there is one large drawback. Both, the mathematical and the experimental evaluation was carried out on non-standard CNNs, where in the pooling-layers neighboring filter responses were combined together by squaring and summing them. The authors called this a "convolutional square-pooling architecture". This is different from the standard max-pooling layers that are nowadays used, where the filter responses are not squared and instead of summing them the maximum response is computed. For this, it remains an open question whether similar results would be achieved with standard CNNs. **Contribution:** Nevertheless, this work suggests that a better architecture can have a larger impact on the the final accuracy of a model compared to the fine-tuning (training) of the filters. With this interesting observation the authors underline that for future works that experimentally evaluate new learning techniques it would be helpful to distinguish the contribution of the architecture from those of the learning techniques by reporting random weight performance as well. A further interesting result was described by the authors: the classification accuracy did not depend on the type of distribution that was used to initialize the random filter weights (at least uniform, Laplacian, and Gaussian distributions were

considered) as long as the distribution was centered around zero.

2017: Convolutional Random Vector Functional Link Network (CRVFL). Zhang and Suganthan [33] demonstrated the benefit of random convolution layers for the visual tracking task. Here a CNN based classifier is learned in order to decide whether image patches contain the image structure to be tracked or not. The authors used not only the original input patch as input for the classifier (FC layer) but also features that were generated with the help of a conv layer where the weights were randomly initialized and fixed. Only the weights to the fully connected layer of the classifier were trained. The results of an experimental evaluation showed that the resulting visual tracking architecture achieves state-of-the-art performance on a visual tracking benchmark. They called this architecture *Convolutional Random Vector Functional Link Network (CRVFL)* since it *"can be regarded as a marriage of the convolutional neural network and random vector functional link network"* where a FC layer also gets as input not only the original input, but a randomized non-linear projection of the input data as well. **Contribution:** Random ConvNets, that map the pixel-representation of an image with the help of a hierarchy of random filters to a feature-representation, can provide a very helpful representation for image classification and visual tracking.

2019: Random feature hierarchies in several different CNN architectures. Rosenfeld and Tsotsos [22] evaluated different standard CNN architectures where the topology was given (fixed), but only a fraction of the filters in the feature hierarchy was learned, while keeping the other filters on their random start values or setting them to zero. For the evaluation they considered wide-residual networks [32], DenseNets [9], AlexNet [16] and VGG19 [26]. The classification accuracy was evaluated on the CIFAR and CIFAR-100 dataset. In the study either only a constant fraction of filters of each convolution layer was trained (fractions considered: 7%, 8%, 9%, 10%, 4%, 70%), or a constant number of filters per layer were trained (1, 5, or 10 filters). The results showed that learning only a small proportion of the filters leads to models with significant higher accuracies compared to random guessing. E.g., for CIFAR-100 a VGG19 where only 70% of the filters were trained (while the other filters were used as they were randomly initialized) achieved an accuracy of ca. 45% (accuracy for random guessing: ca. 1%) and with only 7% of the filters trained the same model architecture achieved an accuracy of ca. 33%. *This work also reported that setting the weights of the filters that were not trained to zero instead of using their random start values, yielded significantly smaller model accuracies*. Unfortunately, the authors did neither provide comparative accuracies when training 100% of the filters in each model/dataset combination nor they evaluated the extreme cases, when training 0% of the filters, i.e., when using all of the filters just with their random start weights in the feature hierarchy. **Contribution:** This important work showed that when more and more of the randomly initialized filters were trained the benefit regarding an improved model accuracy became smaller and smaller.

5 Conclusions

Although these 15 historically significant works have investigated very different architectures in the field of random neural networks and very different experiments with different data sets have been carried out, a common clearly recognizable result remains:

Neuronal networks with (partly) random weights

work surprisingly well. The key to good models lies more in finding or learning suitable architectures than in learning good weights for a specific manually designed architecture.

For future work introducing and experimentally evaluating new neural network architectures and learning algorithms, it should therefore become a standard procedure to also indicate the performance of the proposed architecture when random weights are used. Only this way the contribution of the architecture can be clearly distinguished from the learning algorithm in terms of performance.

References

[1] A. K. Alshamiri, A. Singh, and B. R. Surampudi. Comparative analysis of elm and no-prop algorithms. In *2016 Ninth International Conference on Contemporary Computing (IC3)*, pages 1–5, 2016.

[2] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In David Haussler, editor, *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT'92)*, pages 144–152, Pittsburgh, PA, USA, July 1992. ACM Press.

[3] Weipeng Cao, Xizhao Wang, Zhong Ming, and Jinzhu Gao. A review on neural networks with random weights. *Neurocomputing*, 275:278–287, 2018.

[4] Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. volume 2801, pages 588–597, 09 2003.

[5] Adam Gaier and David Ha. Weight agnostic neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32 (NIPS 2019)*, pages 5364–5378. Curran Associates, Inc., 2019.

[6] Claudio Gallicchio and Alessio Micheli. Deep echo state network (deepesn): A brief survey. *ArXiv*, abs/1712.04323, 2019.

[7] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

[8] Claudio Gallicchio and Simone Scardapane. Deep randomized neural networks. 2020.

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[10] Guang-Bin Huang, Lei Chen, and Chee-Kheong Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *Trans. Neur. Netw.*, 17(4):879–892, July 2006.

[11] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 985–990 vol.2, 2004.

[12] B. Igelnik and Yoh-Han Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6(6):1320–1329, 1995.

[13] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks.

[14] Herbert Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks, 2007.

[15] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision, ICCV 2009*, Proceedings of the IEEE International Conference on Computer Vision, pages 2146–2153, 2009. 12th International Conference on Computer Vision, ICCV 2009 ; Conference date: 29-09-2009 Through 02-10-2009.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[17] Quoc V. Le, Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang W. Koh, and Andrew Y. Ng. Tiled convolutional neural networks. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 1279–1287. Curran Associates, Inc., 2010.

[18] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[19] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[20] Z. K. Malik, A. Hussain, and Q. J. Wu. Multilayered echo state machine: A novel architecture and algorithm. *IEEE Transactions on Cybernetics*, 47(4):946–959, 2017.

[21] Y. . Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[22] Amir Rosenfeld and John K. Tsotsos. Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. In *16th Conference on Computer and Robot Vision*, 2019.

[23] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018.

[24] Andrew M. Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y. Ng. On random weights and unsupervised feature learning. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1089–1096. Omnipress, 2011.

[25] Simone Scardapane and Dianhui Wang. Randomness in neural networks: An overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7, 01 2017.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[27] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100 – 123, 2019.

[28] Fabian Triefenbach, Azarakhsh Jalalvand, Benjamin Schrauwen, and Jean pierre Martens. Phoneme recognition with large hierarchical reservoirs. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2307–2315. Curran Associates, Inc., 2010.

[29] Bernard Widrow, Aaron Greenblatt, Youngsik Kim, and Dookun Park. The no-prop algorithm: A new learning algorithm for multilayer neural networks. *Neural Networks*, 37:182–188, 2013.

[30] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.

[31] Saining Xie, Alexander Kirillov, Ross B. Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1284–1293, 2019.

[32] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC 2016)*, pages 87.1–87.12. BMVA Press, 2016.

[33] L. Zhang and P. N. Suganthan. Visual tracking with convolutional random vector functional link network. *IEEE Transactions on Cybernetics*, 47(10):3243–3253, 2017.