Procedia Computer Science

Volume 88, 2016, Pages 131–138

7th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2016

Design Index for Deep Neural Networks

Prasanna Date, James A. Hendler, and Christopher D. Carothers

Rensselaer Polytechnic Institute, Troy, New York, USA

Abstract

In this paper, we propose a Deep Neural Networks (DNN) Design Index which would aid a DNN designer during the designing phase of DNNs. We study the designing aspect of DNNs from model-specific and data-specific perspectives with focus on three performance metrics: training time, training error and, validation error. We use a simple example to illustrate the significance of the DNN design index. To validate it, we calculate the design indices for four benchmark problems. This is an elementary work aimed at setting a direction for creating design indices pertaining to deep learning.

Keywords: Deep Learning, Deep Neural Networks, Design Index, Performance Evaluation, Neural Network Design

1 Introduction

Deep Learning has proved to be extremely effective in machine learning tasks such as computer vision, natural language processing and voice recognition [8, 14, 15], and although much work has been done in laying the theoretical foundation of DNNs [13] as well as proving its feasibility in specific application areas, designing DNNs seems to be an area which is left relatively untouched in the literature. Although several lines of research pertaining to designing DNNs have been pursued in the literature – for instance, identifying difficulties in DNN training [5, 6], designing hardware to realize the DNN on it [4], various strategies for training DNNs [1, 5, 9, 12], hyperparameter optimization using random search and evolutionary algorithms [2, 3], and selection and creation of even better DNN architectures [10, 16] – only a limited amount of work has been done in investigating the impact of model-specific and data-specific parameters on the performance metrics of DNNs [5, 17].

Performance of a DNN model is greatly affected by both model-specific and data-specific parameters. In this paper, we use three performance metrics: (i) Training Time, (ii) Training Error, and (iii) Validation Error, to evaluate the performance of a DNN model – these are the most widely used performance metrics in the literature. We study the impact of six model-specific parameters: (i) Layer Configuration, (ii) Learning Policy, (iii) Base Learning Rate, (iv) Maximum Training Iterations, (v) Solver Type, and (vi) Batch Size; and three data-specific parameters: (i) Size of Training Dataset, (ii) Noise Percentage, and (iii) Split Ratio.

Selection and peer-review under responsibility of the Scientific Programme Committee of BICA 2016

doi:10.1016/j.procs.2016.07.416

The primary contribution of this paper is the DNN Design Index, which is an index which helps the DNN designer to determine the degree of accuracy and overfitting of the DNN model. We systematically study the process of designing DNNs and conduct DNN design experiments to determine the effect of model-specific and data-specific parameters on the performance metrics of the DNN model for a simple problem of building a DNN that acts as a Weibull classifier. The DNN Design Index is computed for both sets of experiments and it is illustrated how it helps in choosing an optimum configuration. To validate the DNN design index, the indices for four benchmark problems are compared. It must be noted that this is a preliminary work aimed at steering the research community towards creating design indices pertaining to deep learning models.

2 DNN Design Index

The DNN Design Index consists of two indices – the Accuracy Index and the Overfitting Index. The accuracy index gives an estimate of how accurately the DNN was able to learn and the overfitting index gives an estimate of the degree of overfitting that resulted in the learning process. Following sections describe how these indices are derived.

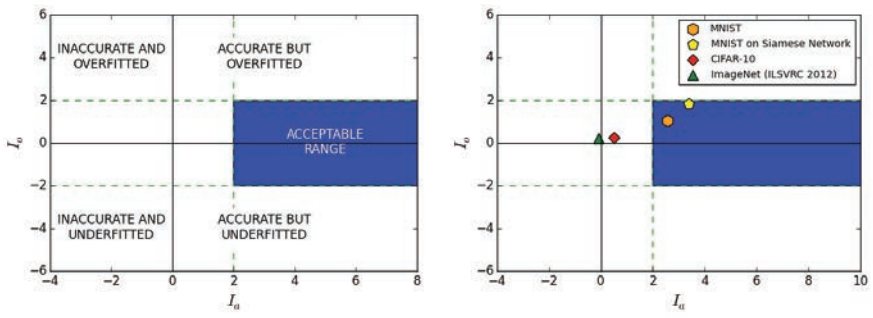
2.1 Accuracy Index (I_a)

To derive the accuracy index, we need to define an entity called threshold error (e_p). The notion of the threshold error (e_p) can be explained as follows. In the training phase of DNNs, the labels or the classes are enumerated, and the training error depends on the order of magnitude of the numeric labels assigned to the classes. While computing the accuracy index I_a, this dependency can be a hindrance. To eliminate this effect of order of magnitudes of the numeric labels on the accuracy index I_a, we define the threshold error e_p. Let p be the order of magnitude of the smallest enumerated label. Then, the threshold error (e_p) can be defined as: $e_p = 10^p$. Given e_p, we want to be able to have a quantitative measure of how well the training error is with respect to the order of magnitude of the labels used during the training phase. Thus, we can define the accuracy index I_a as follows: $I_a = \log_{10}\left(\frac{e_p}{e_t}\right)$; where, e_t is the training error, and e_p is the threshold error.

The accuracy index can be interpreted as follows. If $I_a \geq 0$, then the training error e_t is at most as large as the order of magnitude of the smallest enumerated label. If $I_a < 0$, then the training error e_t is greater than the order of magnitude of the smallest enumerated label. Ideally, we would like to have I_a as large as possible, because the larger it is, the smaller is the training error for a given e_p. Figure 1a illustrates this property. The blue highlighted region shows the acceptable region for I_a, which in this case is greater than two. However, this is not a hard and fast rule. The DNN designer possesses the flexibility of choosing an accuracy threshold T_a for the application specific task such that having $I_a \geq T_a$ would render the model accurate. In this study, we have chosen $T_a = 2$, because for our specific application, we would like to have a training error which is at least two orders of magnitude less than the smallest enumerated label.

2.2 Overfitting Index (I_o)

Overfitting is the phenomenon where the trained model tries to fit the training data to such an extent that it begins to fit all the quirks and random noise in the data instead of modeling the



(a) Acceptable Range (b) Benchmark Problems

Figure 1: Acceptable range of DNN design indices and design indices for benchmark problems

Table 1: Values of each of the six model parameters.

Layer Configuration	Learning Policy	Base Learning Rate	Maximum Training Iterations	Solver Type	Batch Size
3-5-1, 4-6-1	INV, STEP	0.001, 0.01, 0.1	10000, 100000, 1000000	SGD, ADAGRAD, NESTEROV	1, 10, 20

relationship between the variables. Underfitting is the phenomenon where the trained model is too simple to model the data, and would perform bad for both training and validation datasets.

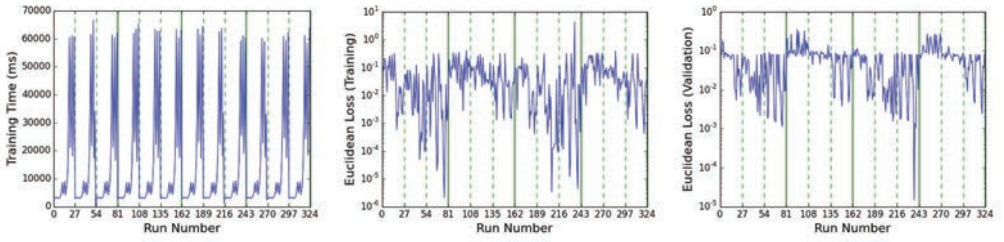
Here, we propose an overfitting index I_o which would help a DNN designer to overcome the problem of overfitting. Given the training and validation errors (e_t and e_v), we need to establish some kind of a metric to measure the deviation between the training and validation error in order to establish a degree of overfitting (and underfitting) of the DNN model. We can define the overfitting index as follows: $I_o = \log_{10}\left(\frac{e_v}{e_t}\right)$ where, e_t is the training error; e_v is the validation error; and, I_o is the overfitting index.

To interpret the overfitting index, we define the overfitting threshold, T_o. The overfitting threshold refers to the number of orders of magnitude that we would like the training error and validation error to differ by in order to render a model as being overfitted or underfitted. As a general rule of thumb and for all computations in this paper, we use an overfitting threshold equal to two, i.e. $T_o = 2$ (i.e. two orders of magnitude less than the validation error).

Given the overfitting threshold, we can interpret the overfitting index I_o as follows. If $I_o > |T_o|$, then the DNN model performs well on training dataset, but not as much as we would like on validation dataset (overfitted) and we must relax it and make it more flexible. If $I_o < -|T_o|$ and the accuracy index I_a is also low, then the DNN model has not learned the training dataset as much as we would like it to learn (underfitted) and we must retrain it to learn the training data set. Ideally, we would like to have an overfitting index in the following range: $-|T_o| \leq I_o \leq |T_o|$. This is shown in figure 1a, where $T_o = 2$. The blue region indicates the acceptable range of overfitting threshold I_o, which is drawn on the y-axis.

Table 2: Values of each of the three data parameters.

Size of Training Dataset	Noise Percentage	Split
100, 1000, 10000, 100000, 1000000, 10000000	5%, 10%, 15%, 20%, 25%, 30%	0.05, 0.11, 0.18, 0.25, 0.33, 0.43



(a) Training Time (b) Training Error (c) Validation Error

Figure 2: Performance metrics for experiment 1.

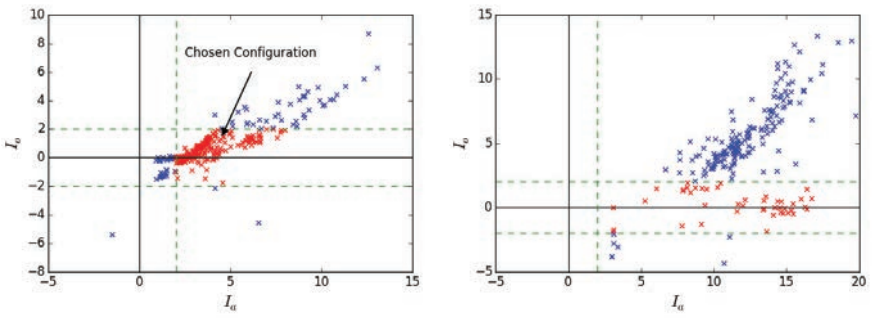
3 Results and Discussion

To study the impact of model-specific and data-specific parameters on the performance of a DNN, we perform two sets of experiments for a simple problem of designing a DNN that can act as a Weibull classifier – i.e. a DNN that would accept data only if it is generated from a Weibull distribution. We choose the Weibull distribution (having shape parameter $k = 0.156$ and scale parameter $\lambda = 15.04$) because we are interested in designing DNNs to detect computer system failures [7]. We focus on six model-specific parameters (Table 1) and three data-specific parameters (Table 2) in the first and second experiments respectively. In both these experiments, we calculate the DNN Design Indices and plot them to illustrate how it can aid in the designing process. We further compute the indices for four benchmark problems.

All the experiments pertaining to the illustrative example of designing a DNN that could act as a Weibull classifier were performed on a machine with two Intel® Xeon® X5650 processors, having six cores each. Each core has a 12 MB Intel® Smart Cache and two threads per core – a total of 24 hyperthreaded cores running at 2.66 GHz. The machine also has 48 GB of RAM and 2.4 TB of RAID storage. The MNIST, CIFAR-10 and ImageNet benchmark experiments were performed on a machine having two Intel® Xeon® E5-2640 v3 processors, running at 2.60 GHz and having 20M cache. Each processor has 8 cores with 2 threads per core – a total of 32 hyperthreaded cores. The machine also had 128 GB RAM and two GPUs: NVIDIA GeForce GTX TITAN Black and GeForce GTX 950. All the DNNs used in the experiments were generated and trained in the Caffe deep learning framework [11].

3.1 Model Parameters

We study six model parameters through 324 experimental runs by analyzing the performance metrics. The results are shown in Figure 2. Figure 2a shows the variation of training time with each experimental configuration. We can see a repeating 'heartbeat-like' pattern of spikes in the figure, which is attributed to the number of training iterations parameter. Each step in the 'heartbeat-like' pattern has a repeating spiking pattern, which is attributed to the batch size values. We see an $\mathcal{O}(n)$ effect of both these parameters on the training time, which is expected. Learning policy and learning rate do not seem to have a clear effect on the training



(a) Experiment 1 (b) Experiment 2

Figure 3: Acceptable runs (marked in red) for experiments 1 and 2.

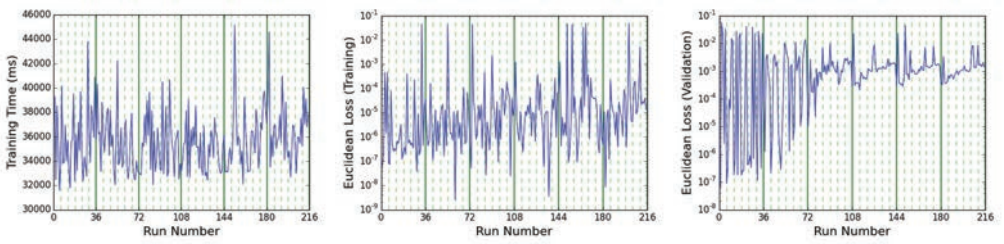
time because the computations affected by these parameters take $\mathcal{O}(1)$ time. The solver type also doesn't seem to have a staggering effect on training time – which essentially means that all the three solvers are comparable to each other. Finally, the layer configuration doesn't affect training time drastically because the two configurations that have been chosen are similar to each other.

Figure 2b shows the variation of training error (computed as the Euclidean loss) with each experimental configuration. The two halves correspond to the 3-5-1 and 4-6-1 configurations respectively, and the two quarters in each half correspond to INV and STEP policies respectively. The INV policy seems to give lower values of training error, but this is true for this particular example and cannot be generalized. Layer configuration does not seem to have a profound effect on training error because the function to be learned is very simple. The base learning rate seems to have a significant influence on training error for the INV learning policy. With a higher base learning rate (0.1), we move away from the initial solution swiftly during initial stages and steadily during the later stages arriving at a better final solution, which is not true in case of lower learning rate (0.001). Maximum training iterations seems to decrease the training error with increasing values as is seen in runs 1-27, 28-54, 55-81. Similar observations can be made for run numbers 163-243. The solver type and the batch size are the two parameters that do not seem to affect the training error as much as the other four parameters. We can see similar results for validation error (Figure 2c).

Figure 3a shows a scatter plot of the two DNN design indices. The DNN model that was finally accepted for the next experiment was run number 239. The configuration of this model was as follows: Layer Configuration: 4-6-1, Learning Policy: INV, Base Learning Rate: 0.1, Maximum Training Iterations: 1000000, Solver: ADAGRAD, and Batch Size: 10. The performance metrics for this model were as follows: Training Time: 34990.67 ms, Training Error: 2.32923E-05, and Validation Error: 1.36776E-03. Using a threshold error $e_p = 1$, and the overfitting threshold $T_o = 2$, the DNN design indices computed were $(I_a, I_o) = (4.63280, 1.76880)$. This model falls in the acceptable range as $I_a = 4.63280 > 2$, and $-2 < I_o = 1.76880 < 2$.

3.2 Data Parameters

We study the performance metrics for three data-specific parameters using six values for each giving a total of 216 experimental runs (see Table 2 and Figure 4). The bold-line partitions in each of the three figures correspond to size of training dataset, the dashed-line partitions correspond to noise percentage and within each dashed-line partition, the six runs correspond



(a) Training Time (b) Training Error (c) Validation Error

Figure 4: Performance metrics for experiment 2.

to the split ratios. In Figure 4a, the training time does not seem to be drastically affected by any of the three data-specific parameters. We do not expect the training time to vary with *data-intrinsic* properties like noise percentage and split ratio. But, the fact that the training time does not seem to be varying with the size of the training data set is counterintuitive. This might be attributed to implementation of Caffe solvers and also the underlying compute infrastructure. It remains out of the scope of this paper to explain in detail the effect of these two factors. However, we can draw a conclusion that when the current implementation of Caffe solvers is run on the compute infrastructure described above, the training time does not seem to be affected by the size of the training data set.

In Figure 4b, the mean training error reaches a minimum to the order of 10^{-5} for size of 100000 and 10^{-3} for all other sizes. This is because for data sets larger than 100000, the DNN model might be overfitting because of less variablity in the training data and for data sets smaller than 100000, the DNN model might be overfitting because of overtraining. This is also supported by Figure 3b, which shows a huge number of configurations in the overfitting range. We do not expect the training error to be affected by the split ratio. In Figure 4c, when the number of data points in the training data set is small, we see a large variation in the validation error caused by the interplay of noise percentage and split ratio. As the size of the training data set decreases, we are more vulnerable to variations in noise and split. As the size of the training data set increases, the validation error increases with increasing values of noise percentage because as the noise percentage increases, the amount of clean data to the trained model is cut down.

3.3 Benchmark Results

We computed the DNN design indices for four benchmark problems: MNIST, MNIST on Siamese Network, CIFAR-10, and ImageNet (ILSVRC 2012). Table 3 describes the configurations of model-specific parameters for these examples. Note that the FIXED policy in CIFAR-10 example simply keeps the learning rate constant throughout the training process. We computed the DNN design indices for all four benchmark problems and plotted them as shown in figure 1b. Both the MNIST tasks are well within the acceptable range. The CIFAR-10 and ImageNet examples are fitted optimally (neither overfitted nor underfitted), but fall short of the accuracy threshold. Recall that the CIFAR-10 and ImageNet models achieved an error of 35.16% in 2009 and 36.7% in 2012 respectively, which, although achievements at the time, simply mean that more than one in a third images would be predicted incorrectly by the models – safe to be called inaccurate in today's world. A DNN designer has the freedom to choose the level of accuracy that would be satisfactory for the specific task – an inherent and essential feature of the DNN design index. If the DNN designers then had a lower accuracy threshold,

Table 3: Network configurations and Design Indices for benchmark problems.

Model-Specific Parameters and Performance Metrics	MNIST	MNIST on Siamese Network	CIFAR-10	ImageNet (ILSVRC 2012)
Layer Configuration	4 Layers, 580 Neurons	10 Layers, 1164 Neurons	4 Layers, 138 Neurons	8 Layers, 10568 Neurons
Learning Policy	INV	INV	FIXED	STEP
Base Learning Rate	0.01	0.01	0.001	0.01
Maximum Training Iterations	10000	50000	60000	450000
Solver Type	SGD	SGD	SGD	SGD
Batch Size	64 Train, 100 Validate	64 Train, 100 Validate	100 Train, 100 Test	256 Train, 50 Test
Training Error	2.59591E-03	3.91600E-04	2.92472E-01	1.14937E+00
Validation Error	2.65981E-02	2.54029E-02	5.28033E-01	1.82050E+00
DNN Design Index	$(\mathbf{2.59}, \mathbf{1.01})$	$(\mathbf{3.41}, \mathbf{1.81})$	$(\mathbf{0.53}, \mathbf{0.25})$	$(-\mathbf{0.06}, \mathbf{0.20})$

then the models would be rendered accurate at the time. A near zero overfitting index and low accuracy index for CIFAR-10 and ImageNet imply that the networks are underfitted, i.e. they are too simple to model the complex datasets. Lastly, we observe that as the datasets get larger and more complicated, the overfitting index gets close to zero.

4 Conclusion

In this paper, we systematically study the influence of model-specific and data-specific parameters on the performance of DNNs. Furthermore, we propose a DNN Design Index which is meant to help a DNN designer in the designing process of DNNs. We illustrate its significance in the designing process by means of a simple example. To validate the DNN design index, we compute it for four deep learning benchmark problems. The DNN design index gives flexibility to the DNN designer in choosing the level of accuracy and overfittedness for the job. It also helps in visualizing how well the DNN performs. Overall, the DNN design index is a tool that helps a DNN designer during the designing phase of DNNs by pointing out the degree of accuracy and overfitting that exists in the model.

Acknowledgement

The research was supported by funding from the Air Force Research Lab (AFRL). We are grateful to Rensselaer Polytechnic Institute for providing the state of the art infrastructure, without which this study would not have been possible.

References

[1] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[3] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[4] Andrew S Cassidy, Paul Merolla, John V Arthur, Steven K Esser, Brian Jackson, Rodrigo Alvarez-Icaza, Piyali Datta, Jun Sawada, Theodore M Wong, Vitaly Feldman, et al. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–10. IEEE, 2013.

[5] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on artificial intelligence and statistics*, pages 153–160, 2009.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[7] Thomas J. Hacker, Fabian Romero, and Christopher D. Carothers. An analysis of clustered failures on large supercomputing systems. *Journal of Parallel and Distributed Computing*, 69(7):652–665, 2009.

[8] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[9] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[10] David Hunter, Hao Yu, Michael S Pukish III, Janusz Kolbusz, and Bogdan M Wilamowski. Selection of proper neural network sizes and architecturesa comparative study. *Industrial Informatics, IEEE Transactions on*, 8(2):228–240, 2012.

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[12] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[14] Yann LeCun, Koray Kavukcuoglu, Clément Farabet, et al. Convolutional networks and applications in vision. In *ISCAS*, pages 253–256, 2010.

[15] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[16] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[17] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.