DEBI-NN: Distance-Encoding Biomorphic-Informational Neural Networks for Minimizing the Number of Trainable Parameters

Supplemental

Appendix A: Harmonized Network Parameters

Both neural networks (NN) and Distance-encoding biomorphic-informational neural networks (DEBI-NN) were built with harmonized parameter sets regarding their basic network configurations (see Manuscript Table 2). Specific parameters for NNs and for DEBI-NNs are in Supplemental Table A1 and Supplemental Table A2 respectively. These parameters were harmonized as much as possible (e.g., activation functions harmonized, early stopping utilized). Note that DEBI-NNs are not trained by gradient descent, but by simple evolutionary principles (Papp et al., 2018; Yang, 2008), therefore, they have specific parameters for their training mechanism. For details see Supplemental C.

Table A1: Parameters of conventional neural networks built in this study. For a comprehensive explanation of the parameters refer to the TensorFlow 2.7.0 documentation: https://www.tensorflow.org/versions/r2.7/api_docs/python/tf

Parameter	Value	Explanation
MODEL		
hidden_neuron_count	100 / 500	Number of neurons in the hidden layer. This study operated with one hidden layer. NNs were trained with 100 hidden neurons in case of tabular datasets, while they were trained with 500 hidden neurons in case of MNIST datasets.
activation	LeakyReLU/Softmax	Activation function. Note that for output neurons Softmax was utilized.
kernel_regularizer	L2	Regularizer function applied to the kernel weights matrix.
optimizer	adam	Name of optimizer
loss	binary_crossentropy	Loss function
TRAINING		
num_epochs	5000	Number of epochs to train the model.
batch_size	min(200, num_samples)	Number of samples per gradient update. Automatically determined by using the minimum value of 200 or num_samples, i.e., the number of training samples in the given dataset.
beta	majority_count/num_samples	Balancing factor. majority_count is the number of training samples in the majority group.

class_weights	{minority_class: beta, majority_class: 1-beta}	Class weights used for weighting the loss function.
validation_split	0.2	Fraction of the training data to be used as validation data if no dedicated validation subset was given.
EARLY STOPPING		
early_stopping	True	Whether to use early stopping or not. Early stopping stops the training when a monitored metric has stopped improving. This study used the validation loss for monitoring.
num_patience	300	Number of epochs with no improvement after which training will be stopped.
min_delta	0.05	Minimum change in the monitored quantity to qualify as an improvement, i.e., an absolute change of less than min_delta, will count as no improvement.

Table A2: Parameters of Distance-encoding biomorphic-informational neural networks built in this study together with their values and explanations. Input neuron counts were specific to the given dataset's feature counts (see Table 2 in Manuscript), while output neuron counts were 2, modeling a binary classification scheme. Note that the table only details the parameter values provided by the user. For additional parameters and how they affect DEBI-NNs see the Doxygen documentation (See Manuscript: Sec. 6)

Parameter	Value	Explanation
MODEL		
Input/Hidden/Output Action Type	LReLu	Equivalent with LReLu activation functions with penalizing negative values by a 0.1 multiplier (Szandała, 2020). Note that the final axon outputs out output neurons were normalized by SoftMax after applying the LReLu action potential calculation.
Input/Hidden/Output Distance Type	GaussianConst	Distances of previous layer axons to current layer somas are mapped by the Gaussian function as of e.q. A1.
Input/Hidden/Output Geometry	Euclidean	Currently the only option implemented. DEBI-NNs are planned to be able to handle non-Euclidean geometries in the future.
Input/Hidden/Output Layer Delta	-0.7	Consecutive spatial neural layers have a 70% overlap in z-direction. A positive number would result in a gap between consecutive spatial layers.
Random Spatial Initialization	true	Generates random spatial x,y,z coordinates for somas and axons of neurons upon initializing an untrained network
Optimize Layer Deltas	false	The network does not include layer deltas in its unknown parameter lists to train; thus, they are kept fixed during the training process.
Hidden Neuron Count	100 / 500	Number of spatial neurons in the hidden layer. This study operated with one hidden layer. DEBI-NNs were trained with 100 hidden neurons in case of tabular datasets, while they were trained with 500 hidden neurons in case of MNIST datasets.
ANALYTICS		
Type	Loss Analytics	Network utilizes principles of loss analytics (Natarajan et al., 2014) to calculate fitness for its model variants during training.
Unit	Entropy Loss	Network utilizes binary cross-entropy loss calculations (Natarajan et al., 2014; Zhang & Sabuncu, 2018). Note that

		this calculation also handles class imbalance by weighting the loss values of samples as of their class imbalance ratios.
Entropy Alpha	0.9	Loss multiplier for correct predictions of a sample. In case the prediction is incorrect, the multiplier for the prediction is 1.0 – Alpha.
OPTIMIZER		
Internal Subsets	true / false	If true: Generates multiple training subsets from the input training set by randomly subsampling it with an 80% sampling ratio from all classification subgroups. Note that for the tabular datasets of this study, this setting was **true**. In case of MNIST datasets, a train-validate-test split was already present, hence, there, this setting was **false**.
Generation Training	Single	The optimizer generates a new population of offspring from the previous population members through performing tournament selection **Population Count** times without saving previous population members to the new population. This means that no elitism is exercised.
Maximum Mutation Rate	1.0	The **Maximum Mutation Rate** is assigned to the first iteration. Over iterations, the **Current Mutation Rate** is decreasing linearly till it reaches the **Minimum Mutation Rate** in the last iteration. In a given iteration, a random number is picked between the -,+ value ranges of the **Current Mutation Rate** to add it to the given unknown parameter of a network. Each unknown parameter gets a new random value within the range of -,+ **Current Mutation Rate**.
Minimum Mutation Rate	0.5	
Mutation Delta	0.5	The unknown parameters of a new offspring are determined by randomly picking parameter entries form the mom and dad member with a 50-50% ratio. The **Mutation Delta** determines the chance that an offspring parameter undergoes an additional mutation afterwards.
Population Count	50 / 100	Number of network variants in the given population. It was 50 for tabular and 100 for MNIST datasets.
Iteration Count	1000	Maximum number of populations the algorithm evolves throughout the training process.
Regenerate Analytics Frequency	10	The number of iterations the model builder keeps the last random split of the training set to perform train and validate steps. Note that in case **Internal Subsets** is false, the input validation subset is used as is, while the training set is still further split to train subsets to minimize the chances of overfitting.
Early Stopping Margin	300	During training a randomly-assigned internal train subset is used to calculate fitness of the parents that are selected to create a new offspring model. The internal validate subset is used to measure the validation fitness of the newly-created offspring afterwards. When a new fittest offspring is identified, training still continues maximum **Early Stopping Margin** times, then terminates. If a new fittest offspring is identified during these additional iterations, the margin counter resets.
Train-Validate Tolerance	0.05	A newly generated offspring model is evaluated by its training and validate fitness. If their absolute difference is less than the Train-Validate Tolerance, the offspring is stored in the **Hall of Fame** container. Note that this tolerance is automatically increased in the first iteration with 0.05 if none of the first population members meet the tolerance criteria. This can happen if the given train-validate subsets are significantly different from each other.
Weights Standardize	true	Input weights are z-score standardized per neuron.

Groups Normalize	false	Group normalization is off.
Model Selection Analytics Count	10	Before the training terminates, the training set is randomly split into the number of subsets and each Hall of Fame member gets the average analytics fitness across the subset splits. In case the training has an input validation subset, this value is not effective.
Model Selection Method	HistoricalTest	Not the offspring from the last iteration are tested to return with the trained model, but the ones stored in the **Hall of Fame** container.
Ensemble Count	1	The fittest offspring from the last population or the **Hall of Fame** population (determined by the **Model Selection Method**) is selected as output of the training. In case this number of higher than 1, the average of the fittest models is generated and returned as the trained model.

$$w_i = e^{-\frac{{d_x}^2+{d_y}^2+{d_z}^2}{GC}} \quad \text{(A1)}$$

where w_i is the calculated weight between the given soma and its i^{th} dendrite (a.k.a. axon of previous layer i^{th} neuron) and d_x, d_y, d_z are the distances of the given soma and its i^{th} dendrite in x, y, z directions respectively. $GC = 9.0$ was chosen as a global constant to normalize distances across the whole network with the same factor.

Appendix B: Relationship of neuron counts and number of unknown parameters to train in NNs and DEBI-NNs

Table B1: Parameter count estimations of neural networks (NN) and Distance-encoding biomorphic-informational neural networks (DEBI-NN) with different neuron count and hidden layer count configurations. I – Input neuron count; H1 – first hidden layer neuron count; H2 – second hidden layer neuron count; H3 – third hidden layer neuron count; O – Output layer neuron count; N – Total neuron count in network; P_{NN} – NN unknown parameter count to train; P_{DBNN} – DEBI-NN unknown parameter count to train; R_P – Ratio of DEBI-NN vs. NN parameter counts to train.

Network configurations					1 hidden layer				2 hidden layers				3 hidden layers			
I	H1	H2	H3	O	N	P_{NN}	P_{DBNN}	R_P	N	P_{NN}	P_{DBNN}	R_P	N	P_{NN}	P_{DBNN}	R_P
2	2	3	5	2	6	8	24	3.00	9	16	42	2.63	14	35	72	2.06
10	6	8	10	2	18	72	72	1.00	26	124	120	0.97	36	208	180	0.87
18	10	14	16	2	30	200	120	0.60	44	348	204	0.59	60	576	300	0.52
26	14	19	21	2	42	392	168	0.43	61	668	282	0.42	82	1071	408	0.38
34	18	24	26	2	54	648	216	0.33	78	1092	360	0.33	104	1720	516	0.30
42	22	30	32	2	66	968	264	0.27	96	1644	444	0.27	128	2608	636	0.24
50	26	35	37	2	78	1352	312	0.23	113	2280	522	0.23	150	3579	744	0.21
58	30	40	42	2	90	1800	360	0.20	130	3020	600	0.20	172	4704	852	0.18
66	34	46	48	2	102	2312	408	0.18	148	3900	684	0.18	196	6112	972	0.16
74	38	51	53	2	114	2888	456	0.16	165	4852	762	0.16	218	7559	1080	0.14
82	42	56	58	2	126	3528	504	0.14	182	5908	840	0.14	240	9160	1188	0.13
90	46	62	64	2	138	4232	552	0.13	200	7116	924	0.13	264	11088	1308	0.12
98	50	67	69	2	150	5000	600	0.12	217	8384	1002	0.12	286	13011	1416	0.11
106	54	72	74	2	162	5832	648	0.11	234	9756	1080	0.11	308	15088	1524	0.10
114	58	78	80	2	174	6728	696	0.10	252	11292	1164	0.10	332	17536	1644	0.09
122	62	83	85	2	186	7688	744	0.10	269	12876	1242	0.10	354	19935	1752	0.09
130	66	88	90	2	198	8712	792	0.09	286	14564	1320	0.09	376	22488	1860	0.08
138	70	94	96	2	210	9800	840	0.09	304	16428	1404	0.09	400	25456	1980	0.08
146	74	99	101	2	222	10952	888	0.08	321	18328	1482	0.08	422	28331	2088	0.07
154	78	104	106	2	234	12168	936	0.08	338	20332	1560	0.08	444	31360	2196	0.07
162	82	110	112	2	246	13448	984	0.07	356	22524	1644	0.07	468	34848	2316	0.07
170	86	115	117	2	258	14792	1032	0.07	373	24740	1722	0.07	490	38199	2424	0.06
178	90	120	122	2	270	16200	1080	0.07	390	27060	1800	0.07	512	41704	2532	0.06
186	94	126	128	2	282	17672	1128	0.06	408	29580	1884	0.06	536	45712	2652	0.06
194	98	131	133	2	294	19208	1176	0.06	425	32112	1962	0.06	558	49539	2760	0.06
202	102	136	138	2	306	20808	1224	0.06	442	34748	2040	0.06	580	53520	2868	0.05
210	106	142	144	2	318	22472	1272	0.06	460	37596	2124	0.06	604	58048	2988	0.05
218	110	147	149	2	330	24200	1320	0.05	477	40444	2202	0.05	626	62351	3096	0.05
226	114	152	154	2	342	25992	1368	0.05	494	43396	2280	0.05	648	66808	3204	0.05
234	118	158	160	2	354	27848	1416	0.05	512	46572	2364	0.05	672	71856	3324	0.05
242	122	163	165	2	366	29768	1464	0.05	529	49736	2442	0.05	694	76635	3432	0.04
250	126	168	170	2	378	31752	1512	0.05	546	53004	2520	0.05	716	81568	3540	0.04

Appendix C: DEBI-NN training

The choice of training scheme

The training scheme is chosen for DEBI-NNs to avoid any inherent bias associated to how weights are trained in conventional NNs. Since the boundary of which state-of-the-art NN training activities alone or in combination may be specific to weight training concepts is unclear, the authors of this work decided at the early phase of their research to rely on training schemes other than gradient descent and all activities related to manipulating weights during training (e.g., dropout, L1 or L2 regularization, etc.). The authors of this work do not state that evolutionary principles are optimal ones to train DEBI-NNs, but consider that the chosen training scheme was adequate to ensure that no weight training-specific practice "leaked" into the concept of training DEBI-NNs. This was necessary to result in a DEBI-NN training approach which can be investigated and better understood within its own capabilities, advantages, disadvantages, and overall, its own merits.

Future work may, therefore, conclude that a DEBI-NN can be trained by methods other than evolutionary principles, however, those investigations shall carefully reflect on the above intention to minimize inherent bias originated from the state-of-the-art.

Note: The current DEBI-NN implementation and Doxygen documentation already contains approaches such as dropout or L1 and L2 regularization. Nevertheless, these approaches were inactive to conduct this study and they are merely there for serving activities of subsequent research.

Basic principles

The principles of training DEBI-NNs are based on evolutionary approaches, relying on modeling natural selection, crossover and mutation (Papp et al., 2018; Yang, 2008). As such, the training algorithm operates with a population of DEBI-NN variants (members) which is evolved across **Iteration Count** times (see Table A2).

DEBI-NN characterization

A population member is characterized by a vector of floating-point values, also referred to as genes that form a digital chromosome. Each gene value describes trainable parameters of the given DEBI-NN (e.g., soma-axon coordinates of spatial neurons). Therefore, a DEBI-NN

model can be instantiated from trainable parameters stored in a digital chromosome (Figure C1) together with model-configuration settings (see Table A2).

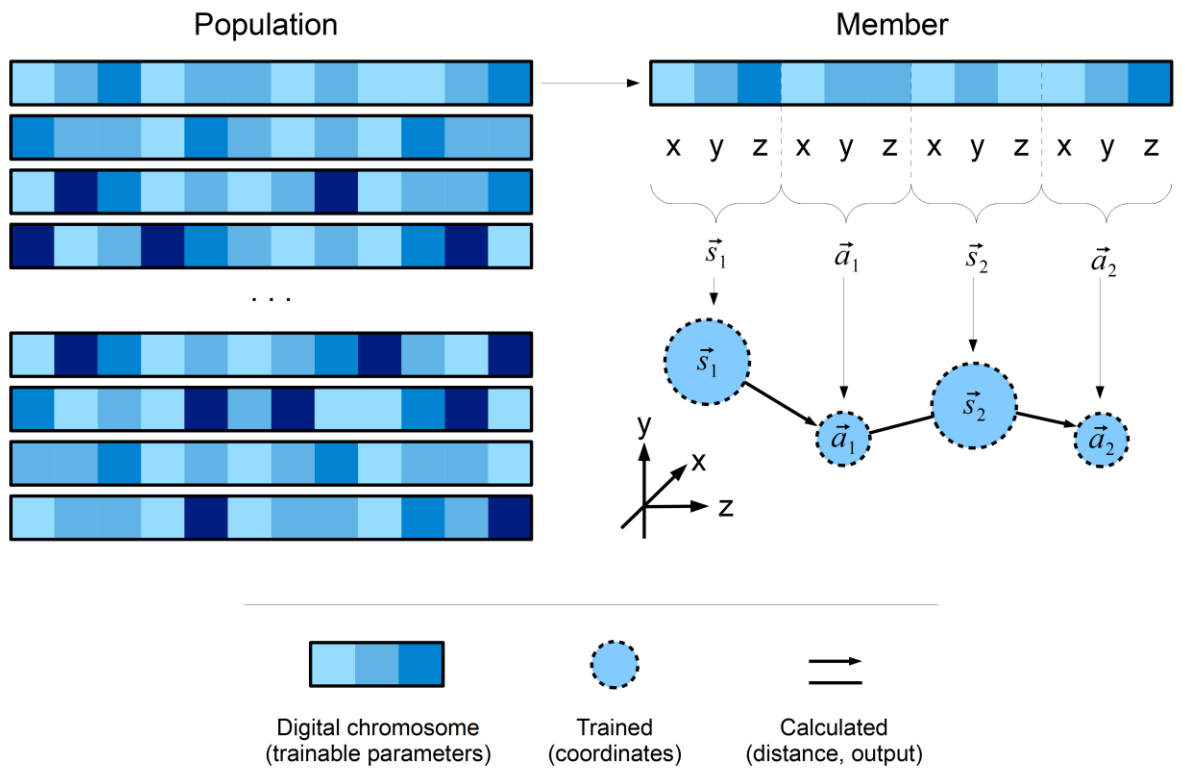

Figure C1: The concept of training DEBI-NN models follows evolutionary principles. The training process operates with iteratively evolving population members. A given member is composed of vectors of floating point values that are x,y,z triplets, describing soma and axon coordinates. In the field of genetic algorithms this vector is referred to as the digital chromosome. In the given example, a 12-long floating point digital chromosome describes 4 x,y,z coordinates of two connected DEBI neurons, each having a soma and an axon pair.

Initialization and iterative phases

The first population contains randomly-initialized DEBI-NN members. Across iterations, the next population members are generated by randomly splitting the current population members to two subgroups and performing tournament selection as of each member's training fitness value (Fang & Li, 2010). The fittest members from each tournament subgroup are selected to be parents of a new offspring for the next iteration. This process is repeated **Population Count** times (see Table A2) to create a new population, which corresponds to a new DEBI-NN training iteration.

Creating offspring

An offspring is created by randomly selecting genes from each parent with a 50-50% chance modeling a crossover of their digital chromosomes. This step is followed by further mutating each offspring gene controlled by **Mutation Delta** and a **Current Mutation Rate** (see Table A2). The final offspring then undergoes a fitness calculation as of a randomly-assigned training subset. In addition, its validation fitness is also calculated to identify, whether the newly-created offspring has a historical best fitness. In case the training and the validation fitness delta is less than **Train-Validate Tolerance**, the offspring is logged in the **Hall of Fame** population container (Figure C2).

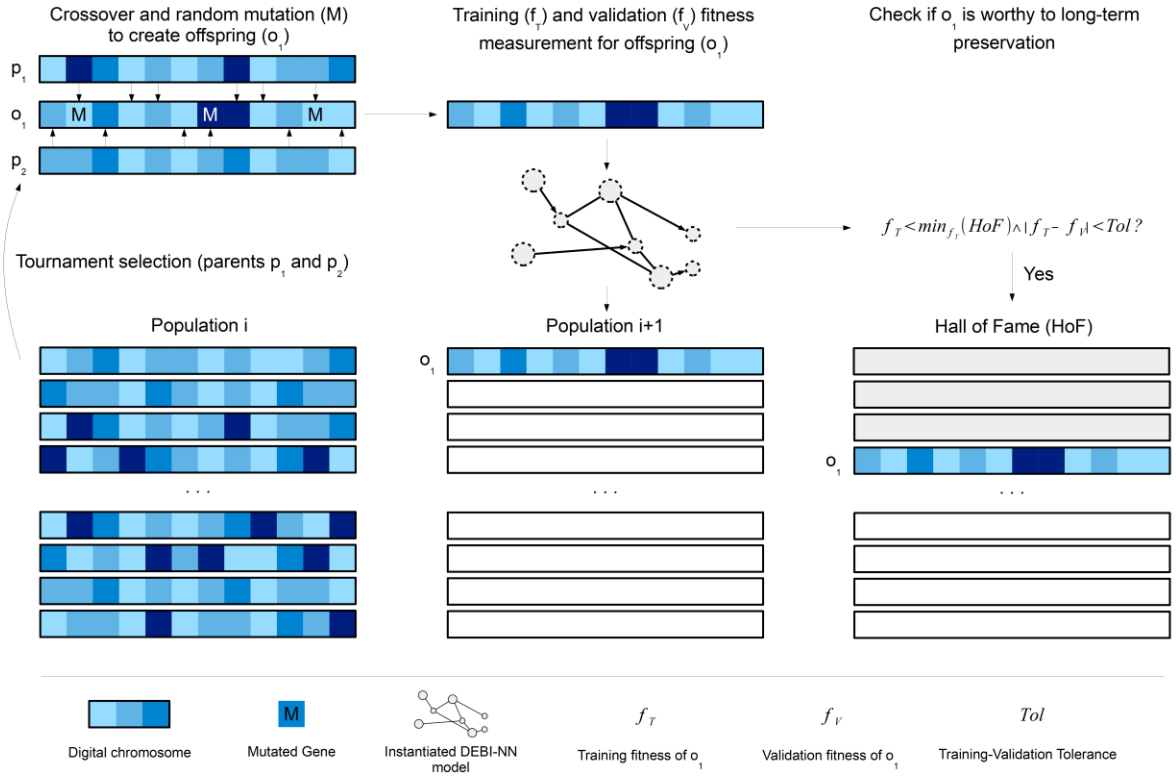

Figure C2: The process of creating and storing a new offspring. Members of the current population (Population i) are randomly shuffled and organized to two subgroups, from which parent 1 (p_1) and parent 2 (p_2) members are selected. A random selection of genes from both parents is performed with a 50-50% chance, followed by randomly mutating selected genes to create an offspring (o_1) The offspring is stored in the new population (Population i+1). The training and validation fitness is calculated for the offspring by instantiating a DEBI-NN model from its digital chromosome as of Figure C1. In case the training fitness is better than the historical best of members stored in the Hall of Fame (HoF) container, and if the train-validate fitness difference is small enough (controlled by the Train-Validate Tolerance – Tol), the offspring is also stored in the Hall of Fame container. The final model or models are selected as outputs of the training from the Hall of Fame container or from the past population container (setting-dependent). Note that the validation fitness is calculated from a validation dataset which is not equal with the test dataset. Latter one is never introduced to the training and it is solely used to estimate the predictive performance of trained DEBI-NN models.

Splitting the training data

The number of randomly-generated training subsets is equal with the number of population members. This is done to optimize training speed on CPU, as each new offspring can be generated and be measured for its fitness in parallel, where the fitness measurement is calculated by a evaluating the assigned training subset of the offspring. The DEBI-NN training process operated with random training subsamples by selecting 80% of samples from each binary subgroup. This was performed to minimize the chances of overfitting by mimicking epoch training principles of conventional NNs.

Fitness calculation

A population member is the fittest with the lowest **Entropy Loss** value, which also considers class imbalance (see Table A2), thus, the DEBI-NN process aims to minimize the loss. It is important to emphasize that evolutionary algorithms do not require training datasets to generate a new offspring, since the fitness of an offspring is calculated once the given offspring was created. Nevertheless, training fitness values are used to select parents during the tournament selection.

Finalizing the training

Fitness values of DEBI-NN members either in **the last population** or in the **Hall of Fame** population are measured by their validation fitness. This step is performed to avoid resulting in a potentially overfitted DEBI-NN specific to any to the training subsets randomly generated for training. The output of the training is a DEBI-NN model whose digital chromosome is the average of the fittest **Ensemble Count** members in the last iteration (see Table A2). This study took the fittest DEBI-NN member as output (**Ensemble Count=1**), thus, did not perform ensemble averaging.

For further details of the implementation see the DEBI-NN Doxygen documentation (see Manuscript: Sec. 6)

Appendix D: Software Packages

For building conventional NNs, TensorFlow 2.7.0 was utilized

https://www.tensorflow.org/versions/r2.7/api_docs/python/tf

Comparison executions were performed using Python 3.9.5

https://www.python.org/downloads/release/python-395/

For hot encoder, feature standardization, and missing data imputation, Scikit-Learn version 1.1.0

https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?ref=https://githubhelp.com

and Numpy version 1.20.2 were employed

https://numpy.org/devdocs/release/1.20.2-notes.html

Feature selection using mRMR (Hanchuan Peng et al., 2005) was performed using the PymRMR package version 0.1.11

https://libraries.io/pypi/pymrmr

Tabular data processing was performed using Pandas version 1.2.4.

https://www.dlr.de/sc/portaldata/15/resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf

References

Fang, Y., & Li, J. (2010). *A Review of Tournament Selection in Genetic Programming* (pp. 181–192). https://doi.org/10.1007/978-3-642-16493-4_19

Hanchuan Peng, Fuhui Long, & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(8), 1226–1238. https://doi.org/10.1109/TPAMI.2005.159

Natarajan, N., Dhillon, I. S., Ravikumar, P., & Tewari, A. (2014). Learning with Noisy Labels. *Advances in Neural Information Processing Systems*, 1196–1204.

Papp, L., Pötsch, N., Grahovac, M., Schmidbauer, V., Woehrer, A., Preusser, M., Mitterhauser, M., Kiesel, B., Wadsak, W., Beyer, T., & others. (2018). Glioma survival prediction with combined analysis of in vivo 11C-MET PET features, ex vivo features, and patient features by supervised machine learning. *Journal of Nuclear Medicine*, *59*(6), 892–899.

Szandała, T. (2020). *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*. https://doi.org/10.1007/978-981-15-5495-7

Yang, S. (2008). Genetic algorithms with Memory- and elitism-based immigrants in dynamic environments. In *Evolutionary Computation* (Vol. 16, Issue 3, pp. 385–416). https://doi.org/10.1162/evco.2008.16.3.385

Zhang, Z., & Sabuncu, M. R. (2018). *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels*. http://arxiv.org/abs/1805.07836