Artificial neural nets

Project 1: Report

Oliver Moravčík

Implemented features

Here I shortly describe functional features included in my implementation.

Multiple hidden layers

Creating a model takes input parameter *dims* as “dimensions” which prescribes overall architecture of feed forward neural net. This parameter is array of values, which of first is number of input values for a single event or record in input dataset and following values are numbers of neurons in each layer. Therefore, if input array is of length 4 as is: *[2, 20, 6, 3]*, that means the neural net will have 2 input values for a single event and 3 layers with 20 neurons in first layer, 6 in second, and 3 in output layer. Implementation of forward and backward feed is independent of number of layers and the net of any desired number of layers and neurons count can be constructed.

k-fold cross validation

I implemented 10-fold cross validation, so for every set of hyperparameters I split the training set to 10 disjunct subsets, and build 10 models as: union 9 of the subsets for estimation set and the complementing subset is validation set. Each model has different validation set. I train all 10 models on their estimation set, use validation set for early stopping and test of the trained model. Then compute mean and minimal error of those 10 models for comparing errors between sets of hyperparameters.

Input processing

I implemented two options of normalization scales for input dataset:

1. ‘abs’ = normalization into interval *<0; 1>* based on maximum and minimum values (absolute scale) which preserves the relative distances:
$$x_i = (x_i - \min(x))/(\max(x) - \min(x))$$
2. ‘std’ = normalization around zero point based on mean value and standard deviation:
$$x_i = (x_i - \mathrm{mean}(x))/std(x)$$

Activation functions

In my implementation I used activation functions provided in source code from the course: Logistic function (also used by name ‘Sigmoid’), Linear function and TanH function.

Output encoding

I used only one-hot encoding provided in source code from the course, because in this classification task I think there is no use or need for different encoding. Classes we ought to predict are not of ordinal character, one-hot encoding provides binary representation and maps outputs in an orthogonal vector space which is better to represent data used in this task.

Training length and early stopping

In my experiments I always used limit on maximal number of epochs at 500, but I implemented early stopping on 3 different terminating conditions on validation error to prevent overfitting and reduce the number of epochs needed to reach satisfying accuracy:

1. ‘*min delay expectancy*’ = number of epochs the model has to reach new minimal error on validation set. If model does not reach lower validation error in specified number of epochs, the training will be terminated.
2. ‘*raised error*’ = percentage of epochs in last *N* epochs, that validation error of the epoch is bigger than validation error of previous epoch. Example: if *N* is 10, and *raised error threshold* is 0.7 (70%), if validation error raises relatively to previous epoch 7 times in last 10 epochs,

the termination condition for training is satisfied and training ends. Values of *N* and *raised error threshold* are input parameters in calling of training procedure.

3. '*accumulated error*' = sum of differences in validation error between the epoch and its previous epoch for specified number *N* of epochs. If the validation error is decreasing with every epoch, the sum is negative (less than zero). If the validation error starts to increase, the sum of errors raises too. If the sum of errors is greater than *accumulated error threshold* the termination condition for training is satisfied and training ends. Values of *N* and *accumulated error threshold* are input parameters in calling of training procedure. The *N* is shared with *raised error*.

In all cases, even when training ends with reaching limit on maximal number of epochs, the training returns weights (and sets the model's weights to weights) from epoch with minimal validation error.

The terminating of training process is allowed only if the model accuracy on validation set is greater than *minimal accuracy* which is also an input parameter.

Weight initialization type

I implemented two options of weight initialization: random values within uniform and normal distribution. Weights can also be scaled into any desired interval (e.g. <0; 1>)

Momentum type

I implemented classic type of momentum.

Error metrics

To measure error, I use only mean squared error (MSE) because it is the most common error metric used in ANN, it penalizes bigger errors more which I think is a wanted characteristic in this task (as opposite to rooted mean squared error (RMSE) for example which is trying to make up for the squaring of errors in MSE).

Model selection

I prepared a set of hyperparameters only slightly different (mostly only by one hyperparameter) and compared results. Then I adjusted set of hyperparameters to follow the settings of the best performing models, optionally added new combination hyperparameters and repeated this workflow.

Subset of examined models

At the end of the document, there is table with some of the models and their results.

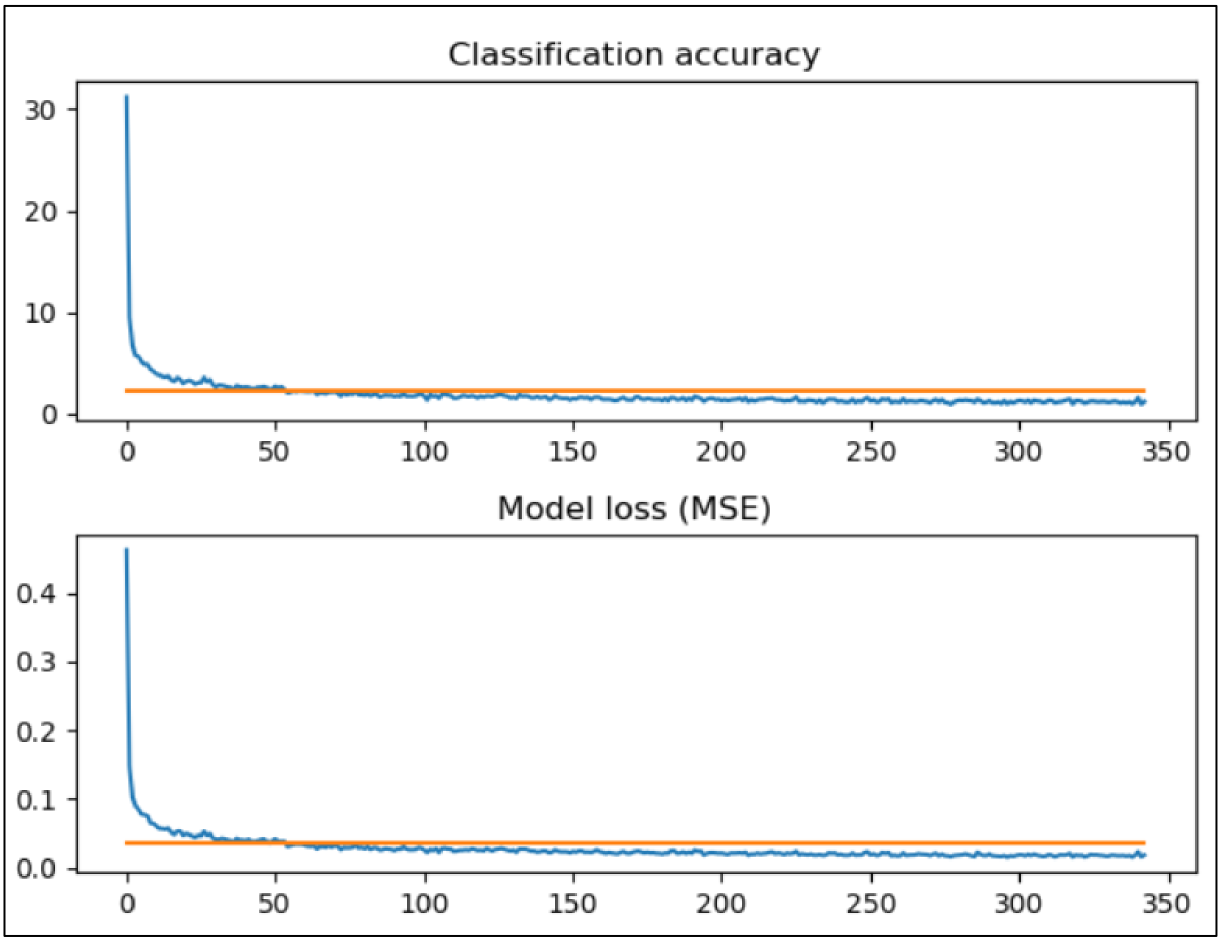
The best model

Hyperparameters

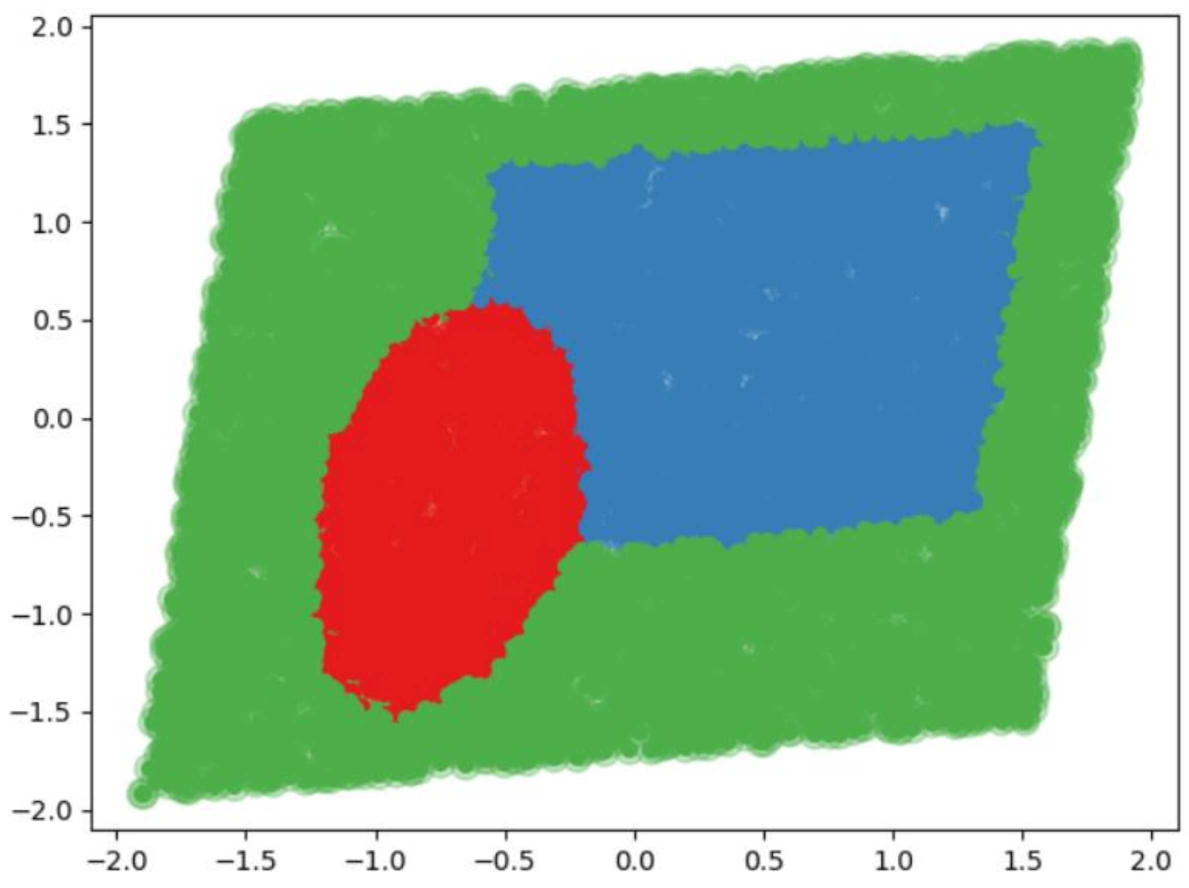
Data normalization type	std (see Input processing section)
Layers architecture	24, 12, 3 (neuron counts on each of 3 layers)
Activation functions	tanh, logistic, logistic (on 3 layers)
Weights initialization	Uniform from 0 to 1
Learning rate (alpha)	0.12
Momentum	0.05

Minimal accuracy for terminating in early stopping	97%
Maximal number of epochs	343
Minimal error delay expectancy in epochs	140
Number of epochs for raised error and accumulated error	30
Raised error threshold	0.66
Accumulated error threshold	0.003

Error vs time



Outputs in 2D



Confusion matrix

Actual (columns) / Predicted (rows)	A	B	C
A	271	9	4
B	0	606	24
C	5	3	1078

data_normalization	layers	functions	distribution	aplha	momen	min_accur	max_epoc	min_delay	q_size	raised_err_	acc_err_thr	mean_valid_	mean_valid_RE	mean_epochs
abs	[2, 20, 3]	['sig', 'sig']	['normal', [-1, 1]]	0.05	0.1	97	500	50	30	0.66	0.0002	0.027	0.048333143	196.1
sdt	[2, 20, 3]	['sig', 'sig']	['normal', [0, 1]]	0.05	0.1	97	500	50	30	0.66	0.0002	0.02775	0.047216137	184.8
abs	[2, 20, 3]	['tanh', 'sig']	['normal', [0, 1]]	0.05	0.1	97	500	50	30	0.66	0.0002	0.0265	0.044322986	127.9
abs	[2, 20, 3]	['tanh', 'lin']	['normal', [0, 1]]	0.05	0.05	97	500	50	30	0.66	0.0002	0.129375	0.234375305	301.3
abs	[2, 20, 6, 3]	['tanh', 'sigmoid', 'sigmoid']	['normal', [0, 1]]	0.05	0.05	97	500	50	30	0.66	0.0002	0.027125	0.043658075	123.7
abs	[2, 20, 6, 3]	['tanh', 'tanh', 'sig']	['normal', [0, 1]]	0.05	0.05	97	500	50	30	0.66	0.0002	0.288625	0.379091823	244.1
abs	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['normal', [0, 1]]	0.05	0.05	97	500	50	30	0.66	0.0002	0.028375	0.044624637	89
abs	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['normal', [0, 1]]	0.05	0.05	97	500	120	30	0.66	0.0035	0.02275	0.032815632	390.6
abs	[2, 20, 10, 3]	['tanh', 'sig', 'lin']	['normal', [0, 1]]	0.05	0.05	97	500	100	30	0.66	0.003	0.02525	0.051193635	345.1
abs	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.05	0.05	97	500	100	30	0.66	0.003	0.025	0.037076922	332.6
abs	[2, 20, 10, 3]	['tanh', 'tanh', 'sig']	['uniform', [0, 1]]	0.05	0.05	97	500	100	30	0.66	0.003	0.033875	0.056888077	385.9
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.05	0.05	97	500	100	30	0.66	0.003	0.013625	0.020388009	244.4
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.05	0.1	97	500	100	30	0.66	0.003	0.014125	0.020620119	262.4
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.05	0.15	97	500	100	30	0.66	0.003	0.015125	0.022163884	263.5
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.05	0.2	97	500	100	30	0.66	0.003	0.015625	0.023384248	232
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.1	0.05	97	500	100	30	0.66	0.003	0.013375	0.019606591	283.9
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.15	0.05	97	500	100	30	0.66	0.003	0.013125	0.019934225	186.8
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.2	0.05	97	500	100	30	0.66	0.003	0.017625	0.02559412	242.8
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	100	30	0.66	0.003	0.010625	0.016375074	225.9
std	[2, 24, 12, 6, 3	['tanh', 'sig', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	100	30	0.66	0.003	0.015625	0.022893158	240.1
std	[2, 24, 12, 6, 3	['tanh', 'sig', 'sig', 'sig']	['uniform', [0, 1]]	0.05	0.05	97	500	100	30	0.66	0.003	0.016875	0.025224936	155.9
std	[2, 12, 6, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	100	30	0.66	0.003	0.019875	0.030022807	225.8
std	[2, 24, 12, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	100	30	0.66	0.003	0.0125	0.018858655	260.1
std	[2, 20, 10, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	100	30	0.66	0.003	0.0145	0.021799417	306
std	[2, 24, 12, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	120	30	0.66	0.003	0.012875	0.019770427	231.9
std	[2, 24, 12, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	140	30	0.66	0.003	0.0115	0.017753121	343.8
std	[2, 24, 12, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	150	30	0.66	0.003	0.016	0.024521562	287.5
std	[2, 24, 12, 3]	['tanh', 'sig', 'sig']	['uniform', [0, 1]]	0.12	0.05	97	500	160	30	0.66	0.003	0.01425	0.020135649	357.4