Machine Learning class (IELE4014): Homework 1 Report

ELKIN CRUZ, UNAL

In this work I present the solution to homework 1 for the machine learning class IELE4014 (Uniandes). I explore all the details and many of the ideas I had while trying to find a good classifier for the given dataset.

1 INTRODUCTION AND DATASET

In the present paper I hope to show the process I followed to try to solve a binary classification problem using two techniques of machine learning, logistic regression and neural networks.

The task is to classify two different genres of music from the MSD genre dataset[1]. The two genres that were intended to be classified for this paper are "dance and electronica" and "jazz and blues", the former with a total of 4935 members and the latter with a total of 4334. The error used in the analysis of the data gattered in the paper is accuracy, or more precisely the "inverse" of accuracy, namely $error = 1 - acc$, and not any other measures like f1-score, recall or sensitivity, to name a few. No measures different to accuracy are required here given that the data is almost evenly distributed over two classes of similar sizes.

The code implementing the training procedures, postprocessing and graphic analysis can be found in https://github.com/helq/haskell-binary-classification.

2 PREPROCESSING

In early attempts, I tried to train the network without preprocessing at all, and the net was not able to learn anything, or at least, nothing fast enough. As it can be seen in the figure 1 the training error never got below 40%, even after 10000 training epochs. The gradient, as can be seen in the figure 2, was very small, but the optimization error (see figure 3) got nontheless small as time went on, suggesting that the net was indeed learning but it was in a region of the optimization space where the gradient was very small.

I decided to normalize each column of the input data using the student's t-statistic[2] and found that the optimization process converged fast to a value around 19%! Take away, normalization is key for solving machine learning problems!

One additional preprocessing step was to apply log to each column on the data and add it as more info to use for learning. The principle behind this preprocessing step was that the correlation between the labels and the data could be non-lineal, especifically, exponential.[3]

[1]The dataset can be found in https://labrosa.ee.columbia.edu/millionsong/blog/11-2-28-deriving-genre-dataset

[2]A detailed description of all possible methods to normalize input data can be found in https://en.wikipedia.org/wiki/Normalization_(statistics). I used the method called student's t-statistic given that I only had the estimated parameters and not the real values for the distributions.

[3]As a side note, applying log to the data gave strange results on training the first time I implemented it, many of the column data became NaN and made the learning process ineffective, the Neural Nets just diverged and couldn't give any prediction. To prevent NaN values on the data I decided to add one (1) before applying log, this little change prevented log to receive 0 as value and allowed me to continue with the training process without much trouble.

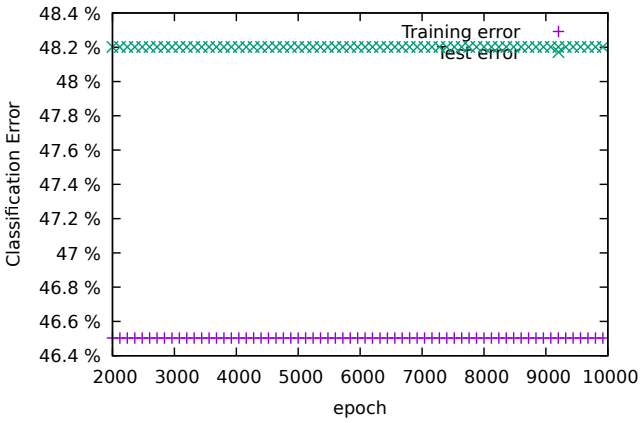

Fig. 1. Training and test errors don't really seem to be improving at all

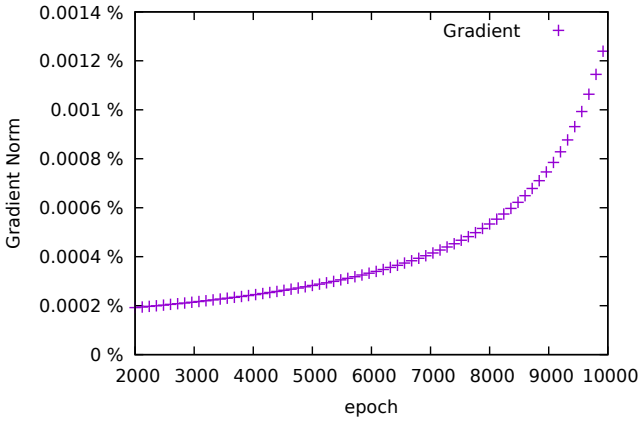

Fig. 2. The gradient of training the NN shows how it is actually learning, but very slowly

The discrete features in the dataset time_signature and key were converted into a one-hot-vector representations. time_signature has 8 (from 0 to 7) possible values while key has 12 (from 0 to 11) possible values. The resulting vector of discrete features for each datapoint has size 21 (8 + 12 + 1, 1 value for the mode binary feature). For each of the 27 continuous features an additional feature was created. The additional features were the result of applying log to each column. A normalization step was performed for each of the 54 continuous resulting features. The total of features used per datapoint was 75 (21 discrete and 54 continuous).

2.1 Training, Testing and Validation datasets

The original 9269 datapoints were distributed randomly in:

- Test set. Size 1390, 15% of 9269

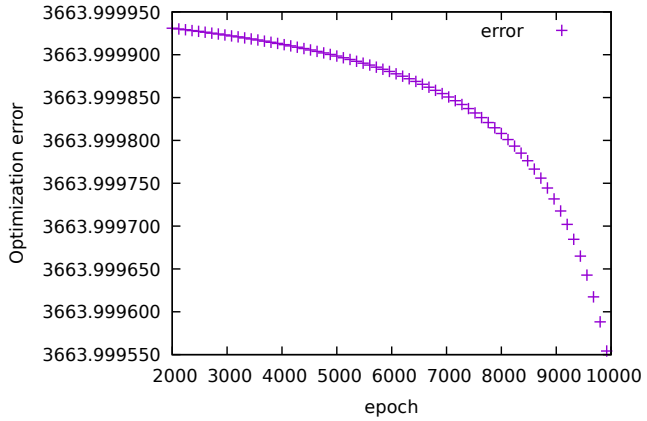

Fig. 3. The optimization/training error decreases with time, but it is painfully slow

- and, Training set. Size 7879, 85% remaining of 9269

The value of 15% was selected as a good compromise between the scare total dapoints for the problem, and a (not so) decent precision error. I refer here about precision in the context of Chernoff bounds. Remember that the additive form of the Chernoff bounds are given by the equation:

$$P\left[\frac{1}{n}\sum_{j=1}^{n} X_j - p \geq \epsilon\right] \leq e^{-2\epsilon^2 n}$$

Given that we only have a dataset, we train only over a train set and not multiple, $n = 1$. X_j is the empirical error we get from the test set. And considering a confidence of $1 - \delta$ ($\delta = e^{-2\epsilon^2 n}$), we can rewrite the equation above as:

$$N \geq \frac{1}{2\epsilon^2} ln\left(\frac{2}{\delta}\right)$$

where N is the size of the test set and δ a value of our choosing.

If we assume $\delta = 5\%$ (a confidence of 95%), we know by solving the equation above that there is at most a ~3.64% difference between the experimental classification error and the real error of the model when the model is tested using the test set, i.e., I know that if I get a 19% classification error on the test set for a model, then this model has a real[4] classification error that lies between 16% and 22%

As shown in the next section three different kinds of models were used to try to solve the problem, the second kind of model used was a neural network with a single hidden layer of arbitrary size. To select the right size on the number of neurons in the hidden network k-fold cross validation was used, for this the training set was divided in blocks of 14% of the dataset size (1297 data points by block). With a size of 14% it was possible to break the training set in six blocks, contrary to use 15% (the value used for the test set) which would only give five blocks to perform k-fold.

An important point to note here is that the same test and training sets, as well as the "validation" sets used in cross validation, were the same for all models tested, i.e., the same original seed was used to randomize the dataset for each test, every test is therefore replicable, all instructions to replicate the results in the current paper can be found in the source code.

[4]When I refer to "real" here I'm talking about all possible datapoints, songs between the two genres, and not only the data that we have access to

3 OPTIMIZATION/TRAINING ALGORITHM

The algorithm used for training all networks in this work was Stochastic Gradient Descent with Momentum (SGDM). SGDM is the default algorithm for optimizing in the grenade library, it's popular enough and gave me reliable results.

In earlier experimentation I found that some of the best parameters to use in training the NNs were:

- Batch size: 100 (around 1/92 of the original set size).
- Stopping conditions:
  1. A maximum of 300 epochs. Any NN that was trained for more than 300 seemed to not improve a lot in the validation test.
  2. Classification Error in Training set (trainCE) gets smaller than 8.5%. Any NN that went below the threshhold of 8.5% was most certaintly memorizing the training data.
  3. Maximum trainCE difference in the last 15 epochs has been smaller than 0.4%, i.e., if after 15 epoch the trainCE didn't change much (all errors are between a margin of 0.4%) then there's no need to continue the training, the NN won't probably learn anything further.

The parameters used for logistic regression were similar to the above, but given the nature of logistic regression (it has a single global minimum) I decided to use all training data at once (no batching), and the third criteria for stopping changed from looking at the last 15 epoch to look the last 30 epochs for a change in the trainCE smaller than 0.4%.

4 MODELS

The homework asked for three different kinds of models to use to solve the classification problem. These models are Logistic Regression, an NN (Multil-layer perceptron) with one hidden layer of arbitrary size, and arbitrarily sized/complex NNs.

The process followed to determine the right model from the two last kinds (NNs with a hidden layer, and arbitray NNs) is detailed in the following subsections:

4.1 Multilayered Perceptron with one hidden layer

The activation function in the output of all hidden layers was $tanh$, and the activation function at the last neuron was the logistic function.

To determine the "right" number of neurons in the hidden layer, k-fold cross validation was performed on the training data. The results of k-fold can be seen in figures 4, 5, and 6.

As it can be seen in figure 4 the cross validation error is smallest for neurons of size 2, 4, and 53 with errors around 16.6% and 16.8%. Remember from "Section Preprocessing" that the real classification error for each model is between -3.7% and +3.7% of the value it appears on the figure, this means that there is no certainty at all

that any of the tested models is better at generalizing that any other by looking at the cross validation errors.

But from the figure 5, it is clear that the models memorize well the training data as more neurons are used in the hidden layer. It must be noted that no model seem to be able to reduce its training error below 6.5%, this is because I decided to stop the training process of any model when it went below 8.5%, the 6.5% threshold is just artificial. If a model with 60 neurons in the hidden layer were to be further trained, ignoring the 8.5% threshold, it will get very low scores. I selected the 8.5% threshold because all NNs that I tried in earlier attempts seemed to not improve after going lower than 8.5% showing symptoms of overfitting (the training error continued to go down while the validation error didn't).

The best models should be those in which the gap between the training error and the (cross) validation error isn't too big. As it can be seen in the figure 5 the best models, those with less overfitting, are the models with 1, 2, 3 and 4 neurons in the hidden layer.

The efect of the 8.5% threshold can be seen in the number of epochs the model was trained before it was stopped, see figure 6. For big models ($neurons > 40$) the effect in the number of epochs necessary to make the NN learn the data is very small, just about 10-12 epochs. The threshold (8.5%) never affects models that cannot memorize enough, those with a small number of neurons ($neurons < 9$) in the hidden layer, the training process is stopped for these models when they stabilize enough.

Models that stabilize and don't seem to memorize much should be considered the best, therefore the best model could either have 2, 3 or 4 neurons in the hidden layer. Given all the above the best model to select seems to be a NN with 2 neurons in the hidden layer.

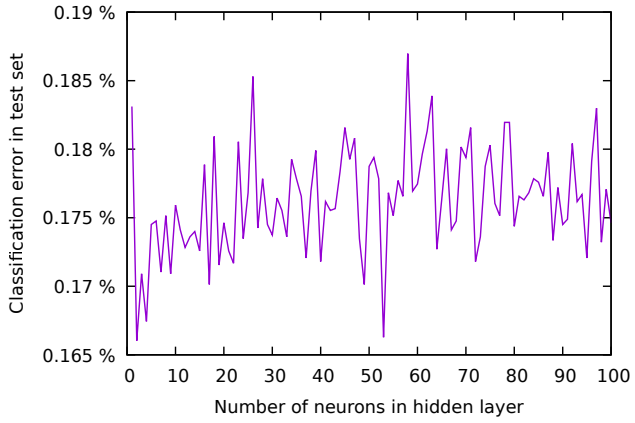

Fig. 4. K-fold Cross Validation Error on number of neurons in hidden layer, no aparent model is better than the rest

4.2 Arbitrary Neural Network

Notice how the best selected model in the previous subsection differs very little from logistic regression, in fact if we take an slightly smaller model, say one single neuron in the hidden layer, then the model would be almost logistic regression (a $tanh$ function is in the middle of the sum of input features and the logistic function).

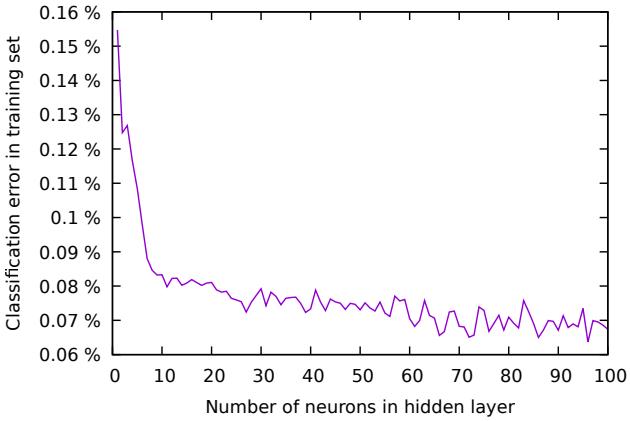

Fig. 5. Mean training error on number of neurons in hidden layer

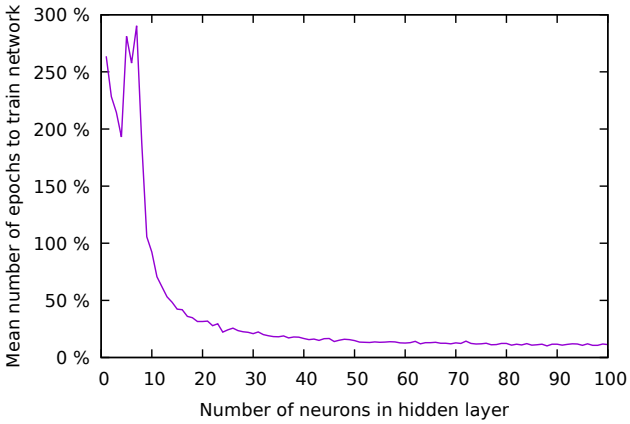

Fig. 6. Number of epochs necessary to train the network. It's big for the NN with little neurons because they can never go below the artificial threshold of 8.5% where any net is stopped, or take a long time to stabilize

The best working NN model found is almost identical to logistic regression, though the NN model seem to perform better than logistic regression (see Section 5) but the error between the two models is still between the error bounds (-3.7%,+3.7%).

I chosed only two models, different to logistic regression and NN with a hidden layer, to show how even different NN models don't seem to get better results than the 2 neurons in the hidden layer network. Proposed models:

- A NN with "two hidden layers": the first layer gets divided into two separate groups of neurons, one learns the data from the discrete features and the other learns the data from the continuous features, see figure for details 7.
- A very huge NN for this problem: three hidden layers with 40, 10 and 3 neurons for hidden layers 1, 2 and 3, respectively.

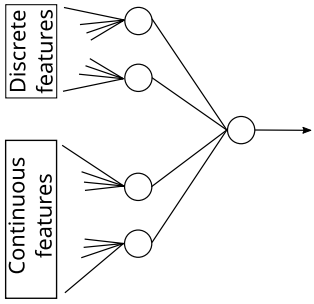

Fig. 7. Special NN, with two hidden layers, one of them is broken into two pieces each one in charge of learning one set of different features, discrete and continuous features

5 TRAINING

5.1 Logistic Regression

Solving logistic regression using Gradient Descent (GD) is very slow, see figure 8, but we can use Stochastic Gradient Descent to find a non-optimal solution quickly and then find the best local minimum using GD, see figure9.

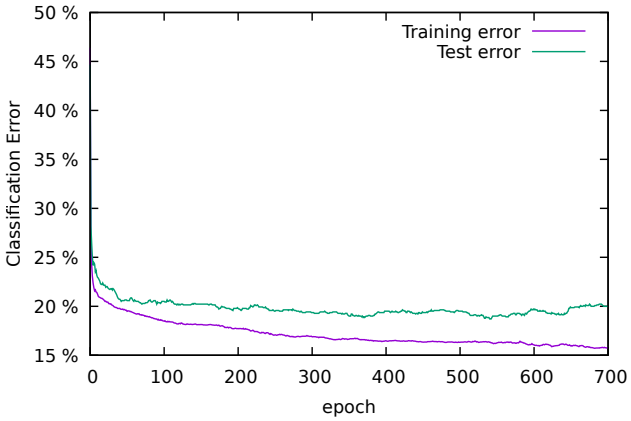

Fig. 8. Solving the logistic problem using Gradient Descent (GD) (all data is used to determine the direction of the gradient per epoch). GD proves to be slow

The lowest training error for logistic regression was ~15.9%, and the test error was ~19.1%.

5.2 NN with a hidden layer of two neurons

The results of training the NN can be seen in the figure 10. The values for the training and test errors are ~12.7% and ~17.6%, respectively.

5.3 Two Arbitrary NN models

The results of training the NN model with two hidden layers, one of which is divided into two different regions (to separate learning of discrete and continous features) can be seen in figure 11. The values for training and test errors are ~13.0% and ~17.7%, respectively.

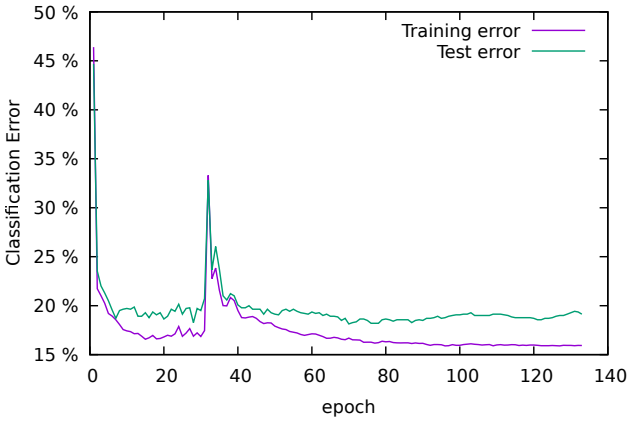

Fig. 9. Solving the logistic problem using Stochastic Gradient Descent (SGD) with an initial batch size of 100, and switching to Gradient Descent (no batching) in epoch 30. Using this two step method is faster to find to a good solution with SGD and improve it to find the best possible solution using GD.

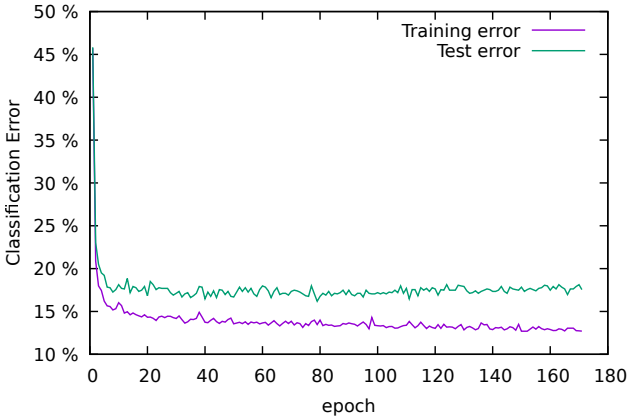

Fig. 10. Training and Test error for a NN with a hidden layer of two neurons.

The results of training the huge NN model (3 hidden layers) can be seen in figure 12. The values for training and test errors are ~7.0% and ~19.9%, respectively.

6 ANALYSIS

Logistic regression gives us a baseline to compare with other approaches. The baseline is therefore around 19.1% classification error using the test set reserved for this purpose.

There is an apparent improvement in the classification error using a Multilayered Neural Net with only two neurons. This more powerful model has an classification error of around 17.6%, but given the small size of the test set (1390 datapoints) and assuming a confidence of 95%, we know that the real classification error lies between $(14.0, 21.2)$ (approximately), which means that there is actually no warranty that this model is in fact better than logistic regression :(.

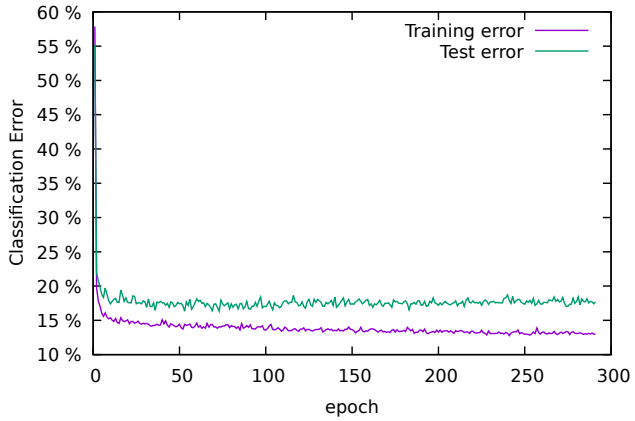

Fig. 11. Training and Test error for the NN that separates the middle layer into two non-conected layers, one for the discrete features and the other continuous features.

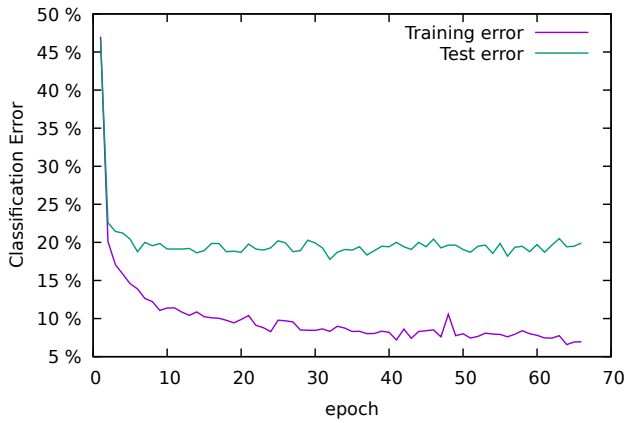

Fig. 12. Training and Test error for the "huge" NN.

A similar conclusion can be derived when analyzing the two additional proposed models. Even though anything I write here about the models could be completely wrong, I could argue that we see how the first model which has a very small number of additional neurons seems to classify as well as the one-hidden-layer 2 neurons NN model. I could also argue that the "huge" model does not perform better than logistic regression, which is a very simple model suggesting that this very huge model is not very good for the task.

The classification error could probably get lower using a better preprocessing strategy. Recall how non-normalized data, no preprocessing on the data, made the process of learning excruciatingly slow.

Another factor that could help improve the classification error is having more data, or finding the right kind of model (this is tough, very hard, because the best model should adjust to the data distribution very closely, but not knowing this distribution is the whole reason of not using analytic methods on this problem).

7 CONCLUSIONS

This homework seemed easy to solve because it isn't that complicated to define a big neural network capable of learing almost any distribution we throw at it, but the very small dimension of the feature space (it was a reduced feature space from other richer feature space, the wave representation of sound, i.e., thousands of datapoints per song and not simply 30 features describing them) and the very small quantity of datapoints/songs available made it incredible hard to minimize the testing error below 17% reliably.

Given the simple classification problem, I presented three-ish models that classify the data with an error of around 17.1%. The problem is hard to solve given the special conditions of it.

There are several aproaches that we could take to try do find a better classifier, but all of them are hard. Either, finding a better preprocessing strategy, having more data available, or finding the right distribution for the data, are hard tasks. Probably the best strategy to follow to improve the classifiers would be finding more data.

Another possible approach is to have access to more features that describe the two classes of datapoints, after all, the 30 features present for each datapoint in the dataset are just a fraction of what the real audio data consists of.

8 APPENDIX

8.1 Details on the source code

All training, testing and validation code was written in Haskell using the library grenade[5], and making an extensive use of dependent types. The library (grenade) is still a work in progress and lacks many standard procedures for training and testing that most libraries for machine learning have. Some of the procedures I implemented for this homework include:

- Training by batches in an epoch (using SGDM)
- K-fold cross validation training
- Arbitrary conditions to stop training process
- Many small procedures and instances from typeclasses definitions to make the above work (e.g., Gradients should be an instance of Num typeclass)

All results presented in this work are replicable, refer to README in the source code for more information on how to replicate the results. To make all results replicable all random numbers used for the creation of the networks and the training procedures depend on the same seed, which wasn't changed while performing the experiments in this work.

As a note aside, working with a new Haskell characteristic (dependent types) was a hard learning process, but proved to be worth it because it was very difficult to write faulty code and compile it. The compiler is very demanding on the correctness of the types up to the point of making it a little hard to work with different types of Neural Networks at the same time (all types are erased on running time), but thanks to the great online Haskell community I could overpass the obstacles[6].

[5]Source code and documentation for grenade can be found at https://github.com/HuwCampbell/grenade

[6]You can follow this question https://stackoverflow.com/questions/46508490/ on how to determine different types of NNs on runtime using dependent types in Haskell