EPFL Machine Learning Higgs Challenge — A Basic Neural Approach

Yuxuan Long, Yiting Zhang, Haojun Zhu
École Polytechnique Fédérale de Lausanne, Switzerland

I. Introduction

The aim of this project is to do binary classification, that requires us to predict background (0) and signal (1). Our first step is to engineer the features, such as feature pruning and data whitening. Then we implement a simple Neural network with three layers. Some modern methods regarding the optimisation are deployed, e.g. Adam update. By ensemble learning, our model achieves an accuracy of 84.4% on the test data.

II. Methods

A. Feature Engineering

Feature engineering aims to process the raw data before feeding them into the algorithms. Here, we mainly want to clean and decorrelate the data, regarding both training and test data.

Feature Pruning: We are only given the training data which has 250000 samples, with labels. For every sample, there are 30 features. Some features are valued as -999, which indicates they are unknown or out of common range. If we feed those "-999" features into the models, they are obviously redundant to the model and may even negatively influence the prediction. Therefore, we remove those "-999" from all the samples. According to the length of pruned data points, by observation we have 6 different data types, as demonstrated in the table below.

Type	Indices for removed features	Quantity	Ratio
A	0 (first element)	4429	0.018
B	4, 5, 6, 12, 26, 27, 28	69982	0.280
AB	0, 4, 5, 6, 12, 26, 27, 28	7562	0.030
BC	4, 5, 6, 12, 26, 27, 28, 23, 24, 25	73790	0.295
ABC	0, 4, 5, 6, 12, 26, 27, 28, 23, 24, 25	26123	0.104
D	None	68114	0.272

Table I: Pruned features and quantity of each data type

From the table, we can deduce that the length of data point ranges from 19 to 30. The right column shows the number of the samples of a specific type in the training data. For the given test data, we have observed that we also only have those 6 data types. So, we can process the test data in the same way as in the training data.
For each data type, we apply a specific model to do prediction. This is reasonable since the samples of different data type may follow some distinct probability distribution.

Data Whitening: After categorising the data into 6 types, we adopt ZCA [1] to perform data whitening, i.e. de-correlation. The basic idea of data whitening is to transform the data so that the covariance becomes an identity matrix. For all samples in the same category, say $\{\mathbf{x}_i\}_{i=1}^N$, we have the covariance $\mathbf{\Sigma} = \frac{1}{N}\sum_i(\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^T$, with the mean μ. Eigen-decomposition gives $\mathbf{\Sigma} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$. By the definition of ZCA, we have the transformation matrix:

$$\mathbf{M} = \mathbf{V}(\mathbf{\Lambda}+\epsilon\mathbf{I})^{-\frac{1}{2}}\mathbf{V}^T \tag{1}$$

where ϵ is a small positive number. The presence of ϵ is for the purpose of increasing numerical stability, especially when the eigenvalues are very small. Given a new sample $\mathbf{x}_{\text{new}}$ in the same data category, we can transform it simply as: $\mathbf{x}_{\text{clean}} = \mathbf{M}(\mathbf{x}_{\text{new}}-\mu)$. Note that the test data points are also whitened using those $\mathbf{M}$ and μ computed from the training data.

B. Models and Methods

As attracted by universal approximation theorem [2], the neural network is used as the main method in the project. For each type of data, a corresponding neural network is trained to make prediction.

Neural Network: We implement a simple version of fully connected neural network, which only has two hidden layers. Given a sample $\mathbf{x}$ of one specific data type, it is then defined as:

$$\mathbf{y} = \mathit{Net}(\mathbf{x}) = \mathbf{W}_3^T\sigma\left(\mathbf{W}_2^T\sigma\left(\mathbf{W}_1^T\mathbf{x}+\mathbf{b}_1\right)+\mathbf{b}_2\right)+\mathbf{b}_3 \tag{2}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are the weights and bias of the network, and $\mathbf{y}\in\mathbb{R}^2$ in our case. $\sigma(.)$ is an activation function known as ReLU [3]. It is simply defined as $\sigma(z)=z$ if $z>0$, otherwise $\sigma(z)=0$. Note that the function $\sigma(.)$ allows element-wise operation.
For the jth input sample $\mathbf{x}^j$, the predicted label can be simply $\mathit{argmax}_{k\in\{0,1\}}\ y_k^j$, where $\mathbf{y}^j = \mathit{Net}(\mathbf{x}^j)$. We use softmax loss to quantify the deviation from true label. If the batch size is N_B, the softmax loss is computed as:

$$L = -\frac{1}{N_B}\sum_j \log\frac{\exp y^j_{gt(j)}}{\sum_k \exp y_k^j} \tag{3}$$

where $gt(j)$ points to the true label for the jth sample.

Back Propagation: We now show how back propagation is derived, especially with a focus on propagating through large feature batch. For the fully connected neural network with l layers, the ith layer is simply modelled as:

$$\mathbf{X}_i = f(\mathbf{Z}_i), \mathbf{Z}_i = \mathbf{W}_i^T\mathbf{X}_{i-1}\oplus\mathbf{b}_i \tag{4}$$

where $f(.)$ can be the activation function or identity function. $\oplus$ here denotes the column-wise addition. $\mathbf{X}$ denotes the input batch such that it collects N_B data points as its columns. At the last layer of the network, we do not have an activation function, i.e. $\mathbf{X}_l = \mathbf{Z}_l$, so $f(.)$ becomes an identity function. Note $\mathbf{X}_0$ is the input data batch to the neural network, e.g. bunch of training samples.
The propagation of gradient starts from last layer. For simplicity of symbols, let $\mathbf{Y} = \mathbf{X}_l$. In our case, while employing the softmax loss, we can derive and construct $\frac{\partial L}{\partial \mathbf{Y}}$ from equation (3):

$$\left(\frac{\partial L}{\partial \mathbf{Y}}\right)_{i,j} = \begin{cases} \frac{1}{N_B}\left(\frac{\exp y_i^j}{\sum_k \exp y_k^j}-1\right), & \text{if } i = gt(j) \\ \frac{\exp y_i^j}{N_B\sum_k \exp y_k^j}, & \text{otherwise} \end{cases} \tag{5}$$

Note that $\mathbf{y}^j$ is the jth column of $\mathbf{Y}$ at this case.
Assume that from the gradient computation at the deeper layers, we

already know $\frac{\partial L}{\partial \mathbf{X}_i}$ and let ReLU be the only activation function. Since the derivative of ReLU $\sigma(z)$ is rather simple, for $i < l$ we have: $\frac{\partial L}{\partial \mathbf{Z}_i} = H(\mathbf{Z}_i) \odot \frac{\partial L}{\partial \mathbf{X}_i}$, where $H(.)$ is the element-wise heavy-side function and $\odot$ denotes Hadamard product. By some basic matrix calculus, from the equation (4) we can further derive:

$$\frac{\partial L}{\partial \mathbf{W}_i} = \mathbf{X}_{i-1} \left(\frac{\partial L}{\partial \mathbf{Z}_i} \right)^T, \frac{\partial L}{\partial \mathbf{b}_i} = \frac{\partial L}{\partial \mathbf{Z}_i} \mathbf{1}, \qquad (6)$$

where $\mathbf{1} \in \mathbb{R}^{N_B}$ is a vector of ones. We also simply derive that $\frac{\partial L}{\partial \mathbf{X}_{i-1}} = \mathbf{W}_i \frac{\partial L}{\partial \mathbf{Z}_i}$, which can be used to compute the gradients for the previous layer, e.g. $\frac{\partial L}{\partial \mathbf{W}_{i-1}}$ and $\frac{\partial L}{\partial \mathbf{b}_{i-1}}$. This is nothing other than applying chain rule in a backward manner

Optimisation: We use the mini-batch stochastic gradient descent to minimise the loss L. In addition to this, $l2$ regularisation is applied to all parameters, so that the weights are decayed in each update during the optimisation. Adam update rule [4] is also used to accelerate and smooth such non-convex optimisation. As ReLU is used throughout our network, we adopt He's method [5] to initialise the weights $\mathbf{W}_i$ as the Gaussian noise $\mathcal{N}(0, \frac{2}{d})$, where d is the number of rows in $\mathbf{W}_i$.

C. Model Selection and Ensemble Learning

In order to have an unbiased test, K-fold cross validation is used. In this way, we train the model using $K - 1$ folds, and test it on the other fold. We can obtain K different models from one specific machine learning method. Besides this, those models can also vote for the final prediction, which may boost a bit the performance. K should be odd in order to make a clear final prediction.

III. Results

A. Experiment with Neural Network

For data whitening, we choose $\epsilon = 10^{-9}$. About the architecture of our neural network, for the data type A and AB which have limited number of samples, the first hidden layer has 25 neurons and the second hidden layer has 10 neurons. For the other data types, as inspired by [6], both hidden layers have $n + 4$ neurons, where n is the dimension of the input data. During the stochastic gradient descent, the learning rate is set to 10^{-4}, the weight decay is 0.001, the batch size is 100. In the experiment, we only run 300 epochs, due to the consideration of computational cost and overfitting issue. To compare the performance, we also implement ridge regression and logistic regression. For logistic regression, we set number of iterations to be 100, where full size is taken using the Newton's method. The $l2$ regularisation parameter is 10^{-5} for logistic regression, and 10^{-3} for ridge regression.

B. Performance Evaluation

Classification accuracy is used as the main evaluation metric in this project. By applying K-fold cross validation, we divide the given training data into $K = 5$ folds. For each fold, we train 6 neural nets respectively for those 6 data types. The model trained on a specific type of data has a distinct accuracy, an example is shown in table III. The overall performance combining all data types is here estimated as the weighted average accuracy, where the weight is the ratio of the number of samples with a specific type to the total number, as shown on the rightmost column of the table I. For example, the data types B, BC and D dominate in the data i.e. each occupies over 25%, so they contribute a lot to the final performance. The weighted average accuracy is measured across each method and the test fold, as shown in the table II. It is not surprising that our neural network outperforms than the basic methods.

With a typical focus on the fold 1, we can see that the neural net model trained on data type A, AB, ABC achieves an accuracy above 0.9. This does not mean a good model. In the table III, recall and precision are also computed. It is easy to observe that the recall is lower than the precision. For the data type A, AB, ABC, the models have a typically low recall, even below 0.5. This means there are numerous false negatives, so that the models tend to conclude that any samples are negative. Such "picky" behaviour can be explained by the imbalanced proportion of positive and negative samples in the training data. For example, negative samples in data type ABC account for 94%. For the data type B and D, the training samples are more balanced, so that their recalls are much higher than the other. However, the models trained on them do not have a high accuracy, e.g. below 0.85. This can be explained in figure 1, that the softmax loss does not converge to a small value, particularly for the data type B, BC and D. This drags the mean accuracy to 84%. Fortunately, for each data type, 5 neural networks vote for the final prediction, which increases the accuracy to 84.4% on the test data.

Fold / Method	1	2	3	4	5	Mean
Ridge Regression	0.7654	0.7650	0.7671	0.7640	0.7661	0.7655
Logistic Regression	0.6632	0.7325	0.7336	0.7290	0.7100	0.7137
Neural Network	**0.8400**	**0.8415**	**0.8410**	**0.8408**	**0.8399**	**0.8406**

Table II: Weighted average accuracy of each method on test folds

Data type	A	B	AB	BC	ABC	D
Accuracy	0.923	0.805	0.911	0.820	0.952	0.837
Recall	0.581	0.715	0.187	0.675	0.281	0.821
Precision	0.718	0.760	0.700	0.749	0.720	0.829

Table III: An example of performance evaluation on fold 1

IV. Discussion

Although the result of neural network beats other basic machine learning methods, there are some unsolved issues. First, the loss always fluctuates during optimisation, which may be due to the high randomness of the data information or the improper design of gradient descent. This can influence the convergence. A better design of the optimisation algorithm is expected. Second, the negative and positive samples are not balanced. There are many tricks to solve this, e.g. add penalty on giving the false negatives. Some explorations on hyper-parameters are worth to try, like increase the width of the network. However, enlarging the network may bring considerable computational cost. Feature augmentation is also worth to try, since it may make more convenience for the model to interpret the latent information, e.g. a better decision boundary. The network architecture can be also modified. Besides increasing the depth, we can add batch normalization as to adaptively feed a better distribution into the activation function.

References

[1] A. Kessy, A. Lewin, and K. Strimmer, "Optimal whitening and decorrelation," *The American Statistician*, vol. 72, no. 4, pp. 309–314, 2018.

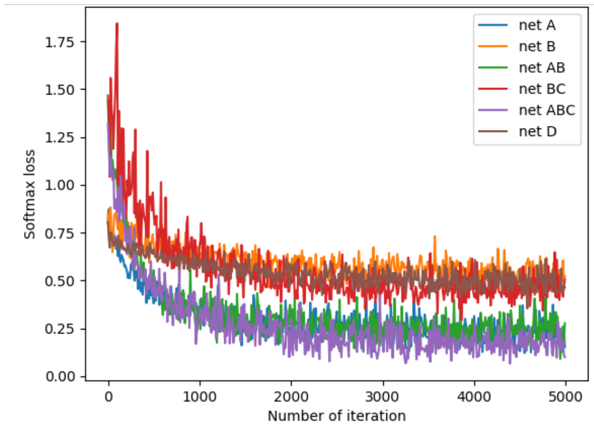

Figure 1: Softmax loss for the models trained for 6 data types

[2] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.

[3] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[6] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in neural information processing systems*, 2017, pp. 6231–6239.

V. Appendix

Several basic computing models are used to find weights $\mathbf{W}$ which can minimise the loss function for regression. Namely, they are least squares, ridge regression and logistic regression.

Least Squares: Least squares' concept is to minimise the mean-square loss function. As we know that mean-square loss function is

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - \mathbf{x}_n^T\mathbf{w})^2 = \frac{1}{2N}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w}) \quad (7)$$

It needs two requirements to find the minimum of the loss. Firstly the loss function is required to be convex, and secondly, the gradient of the function is zero. Then we have

$$\mathbf{w}^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (8)$$

Thus, it can be used to predict new test datapoint $\mathbf{x_m}$:

$$\hat{y}_m := \mathbf{x}_m^T\mathbf{w}^\star = \mathbf{x}_m^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (9)$$

Ridge Regression: While least square model may cause the problem of overfitting which is undesirable, regularisation is added. Euclidean norm is used as regulariser and we have ridge regression:

$$\min_{\mathbf{w}} = \frac{1}{2N}\sum_{n=1}^{N}(y_n - \mathbf{x}_n^T\mathbf{w})^2 + \lambda||\mathbf{w}||_2^2 \quad (10)$$

Then we get:

$$\mathbf{w}^\star_{\mathbf{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda'\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \quad (11)$$

which can be used for the further prediction of new test data. It can be noticed that ridge regression will be least squares model when $\lambda = 0$.