Creating a machine learning model to separate a Higgs decay channel from background events

Mattias Hollen Aasen

September 20, 2022

Abstract

In this project we develop a supervised machine learning model that will attempt to solve the classification problem of differentiating between a signal and background event using data from the Higgs Machine Learning Challenge from 2014 as test and training sets. Our model achieved an accuracy of 0.837 ± 0.003 and a loss of 0.363 ± 0.004 on the training set, and an accuracy of 0.624 ± 0.003 and a loss of 0.810 ± 0.004 on the test set.

1 Introduction

The 2012 discovery of the Higgs boson at the LHC was a significant milestone for the field of physics, as the Higgs boson was the only particle from the Standard model that had not been observed prior to this discovery. The observation of the Higgs particle is done not by observing the particle itself, but by observing the particle it decays into. The observation of a specific decay or particle collision is referred to as an event. At the LHC there are up to a billion events taking place every second, and so we will need effective computational methods to separate the events we are interested in from the others. The field of machine learning offers a great number of possibilities for solving this task.

In this project we will create a supervised machine learning model that will attempt to solve a binary categorization problem. We will base our problem on the Higgs Machine Learning Challenge from 2014. The goal of the challenge was to improve the procedure of creating targets for the test data. By using the basic method provided by this challenge to create labels for the events we get access to a training and a test set that simulates particle collisions at CERN. Each event contains 30 parameters that describes different physical parameters for a particle collision at the LHC. A signal in this dataset corresponds to a Higgs particle decaying into two tau leptons. A background event corresponds to any of the other possible particle interactions that is observed at the LHC.

The data sets have a few attributes that needs to be adressed in order for a machine learning model to be able to work with them. Preprocessing the data sets will therefore be the first part of this project. Further we will be generating the labels for the test set using the AMS method provided by the Higgs ML challenge. Finally we will create a model that can differentiate between signal and background events.

2 Theory

2.1 Supervised Machine Learning

In this project we aim to train a feedforward neural network into acting as a binary classifier [2]. We want our network to approximate a binary classifier $y = f^*(x)$ that maps the input x to output y through a function f^*. The neural network will approximate this through the mapping $y = f(x;\theta)$ where θ indicates some parameters of the network that leads f to the best possible approximation of f^*. The goal of training the network is to find the parameters θ that make this approximation as close as possible. This is done by using a training set which provides the network with examples of features, which forms the input x, along with corresponding labels, which is the desired output y.
It is common for neural networks to consist of more than one function, called layers. As the network goes through a training cycle, often referred to as an epoch, it will update the parameters in each of the layers in order to steer the model closer to producing the desired output. The number of layers, the width of these and the number of epochs are examples of what is called hyperparameters. These are parameters set before the network starts training and which remains untouched once training has begun.

In order to measure how well the network performs we can look at the accuracy or the error rate of the model, which tells us the proportion of examples that produce a correct or incorrect output, respectively [3]. This is done by implementing a cost function. One way of minimizing the cost function is to use an optimization technique called stochastic gradient descent. For a cross-entropic loss function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta) \quad ,$$

where $L(x^{(i)}, y^{(i)}, \theta) = -\log p(y|x;\theta)$ is the per-example loss, the gradient $\mathbf{g} = \boldsymbol{\nabla}_\theta J(\theta)$ can be estimated as

$$\mathbf{g} = \frac{1}{m'}\boldsymbol{\nabla}_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta) \quad .$$

Here m indicates the total number of examples provided in the input, while m' is the minibatch size. Instead of using every example provided in the input the algorithm will on each step create a minibatch drawn from the training set, for which the purpose is to reduce the computation cost and training time.
The parameters θ are then updated by using the gradient,

$$\theta \leftarrow \theta - \epsilon g \quad ,$$

where ϵ is the learning rate. It determines the size of each step the algorithm take as it attempts to move to the minimum of the loss function.
If the model is not able to obtain a sufficiently low training error we say it is underfitting. If the model is not able to obtain a satisfyingly small gap between test and training error it is overfitting. The capacity describes how complex a model is. The larger it is, the better its ability to fit functions will be. If the capacity is too large the model will be prone to overfitting, as it will fit functions too well to the training data instead of generalizing enough to be applicable to real data. Low capacity may lead to issues with fitting the training set, leading to the model being too general.

To avoid underfitting we can observe what choice of hyperparameters, which determines the capacity of the model, that leads to the smallest training error. To avoid overfitting we use a validation set which we draw from the training set. This is because we should not make changes to the hyperparameters based on how well the model performs on the test set, otherwise the point of having a test set vanishes. If there is too little test data to create a sufficiently large validation set it can cause the test error to have a large uncertainty, making it difficult to find the optimal choice for the hyperparameters. This can be solved by cross-validating the data, where the validation data is drawn from different parts of the total test set across a certain number of times and the test error is estimated for each time. The average test error is then calculated based on the different test errors.

2.2 AMS

The Approximate Median Significance (AMS) is a function used to generate labels for the test data in the Higgs Machine Learning Challenge [1].
The AMS is described as

$$\text{AMS} = \sqrt{2((s+b+b_r)log(1+\frac{s}{b+b_r})-s)} \tag{1}$$

where b_r is the constant regularization term, s is the unnormalized true positive and b is the unnormalized false positive.

3 Methods

All of the code used for this project can be found at github.

3.1 Data generation

The data used for this project is provided by the Higgs ML Challenge from 2014. It consists of a training set containing 250 000 events and a test set containing 550 000 events. For the training set, 85667 of the events are labelled as signals, while for the test set the number of signals is 273884. Each event consists of 30 parameters, representing some physical quantity. The events in the training have an additional parameter for labels and for weights, which the test data does not have. The label is either "s" if the event is a signal or "b" if it is a background event. Some variables in the data sets are undefined, which is implied by a numerical value equal to -999.

3.2 Preprocessing

Before a machine learning model can be created the data must be processed. The first issue is the labels being given as strings. We solve this by changing the background events "b" to be 0 and the signal events "s" to be 1.
The variables of the different parameters have a magnitude between 10^{-1} to 10^{2}. As we want the output to be either 0 or 1 this can create issues with the weights in the layers of the model,

potentially causing the computing time to be longer or decreasing the accuracy of the model. In order to remove this potential source of error and processing time-drain we normalize each feature to be on the same scale as the output, ranging from 0 to 1.
Ignoring the -999 values, the lowest value for each feature is set to 0 by subtracting this value from every entry in the corresponding column. The undefined values are then manually set to be 0.
The training data is divided into ten sets and ten folders are created. Each folder has a validation set given by one of the ten sets and a training set made of the remaining nine sets. The set used as a validation set varies in each folder, giving ten unique combinations of validation and training data.

3.3 AMS

The test data we are provided with is given without the necessary labels for testing our model. As suggested by the Higgs ML Challenge we will create these labels by using the Approximate Median Significance (AMS) defined in equation (1). We do this by implementing the code from the starting kit provided by the Higgs ML Challenge. The starting kit is a tool which creates a file containing the event ID, rank order and label for each event found in the test set. The label column is extracted into a single array to be used for testing the model together with the test features.

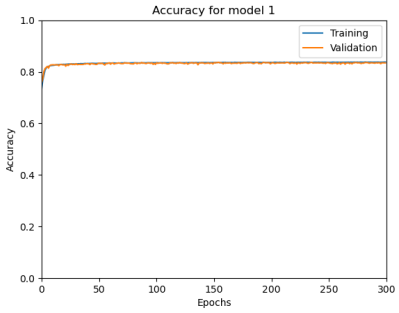
3.4 Machine learning model

We define three models with different choice of layers to test the validation set with. Our first model consists of 3 layers with 10 nodes each. For the second model we use 5 layers with 50 nodes each. The third model has 3 layers with 512 nodes each. It also uses a dropout function and a regularizer between each layer in an attempt to reduce the chance of the model overfitting. We compare how each model perform when tested on one validation set and progress with the most promising choice of hyperparameters. After trying out different optimizers and loss functions we ended up using binary cross-entropy as our loss function and Adam as the optimizing function. Changing the learning rate did not seem to improve the performance so we leave it at the default value of $\epsilon = 0.001$.
Finally we see how our model performs using the test set provided by the Higgs ML Challenge. We use 100 epochs, as any more did not seem to provide any improvement to our model. Given that our model stabilized after approximately 20 epochs we could have reduced it further, but we were curious to see if it stabilizes after the same amount of epochs or if more would be needed.

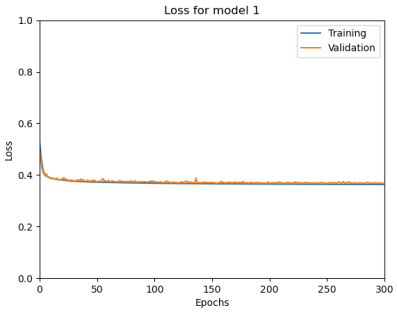
4 Results

In figure 1a and 1b we see the accuracy and loss for the first model, respectively. The model performs great on the validation set with a low error for both training and generalization. We see that both the accuracy and loss values converges after a short amount of epochs, and the model does not seem to improve significantly during the remaining epochs. The final values can be seen in table 1 along with the results from the other models.

In figure 2a and 2b we see how the second model performed. The training error is lower for the accuracy than for the first model. The generalization error is somewhat larger but still very low.



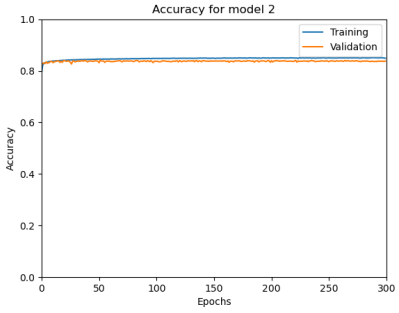
(a) Accuracy for the first model.



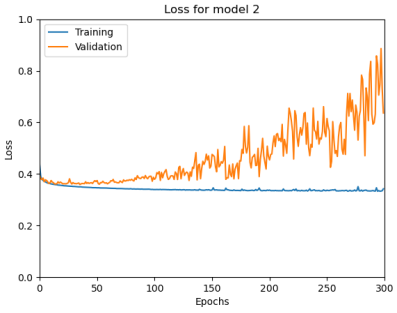
(b) Loss for the first model.

Figure 1: Results for the first model.

We see that the loss for the training set is lower than for the first model, but the loss for the validation set grows larger at later epochs. This shows us that our second model is overfitting, and using additional epochs does not seem to improve this. We also see that the loss and accuracy are not directly correlated. The difference in loss and accuracy could be due to how our training set is made up. If the training set consists mostly of background events and our model predicts every event to be background, a high accuracy would still be achieved. It might be more relevant to use other metrics in order to assess how well the model performs, in that case.



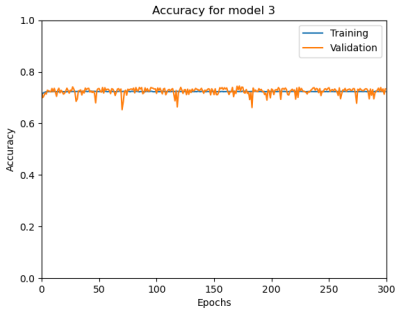
(a) Accuracy for the second model.



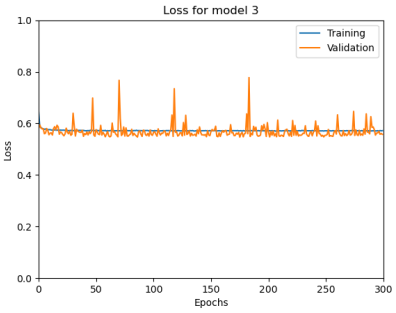
(b) Loss for the second model.

Figure 2: Results for the second model.

In figure 3a and 3b we see how the most last model performs. The accuracy is lower than for the two previous models. The loss for both validation and training is higher than for the first model, but the generalization error seems to be low. The loss values for the validation set seems to spike significantly at some epochs. This might be solved by building the model using more epochs. Seeing as we have a simpler model that requires less epochs to stabilize, this is not something we choose to look further into in this project.



(a) Accuracy for the third model.



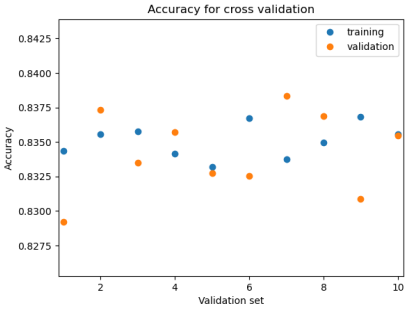
(b) Loss for the third model.

Figure 3: Results for the third model.

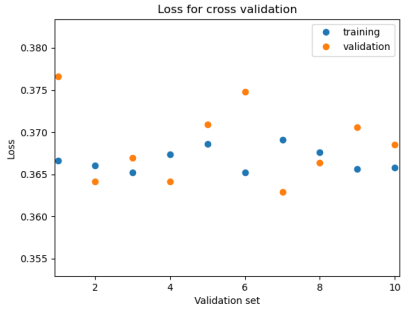
Table 1: Loss and accuracy for each model.

	Training accuracy	**Validation accuracy**	**Training loss**	**Validation loss**
Model 1	0.838	0.834	0.363	0.367
Model 2	0.849	0.837	0.342	0.636
Model 3	0.723	0.734	0.572	0.557

In figure 4a and 4b we see the result of cross-validating ten different batches of the training set individually using our first model. The mean error for accuracy ended up being 0.003, while for loss the mean error was 0.004.



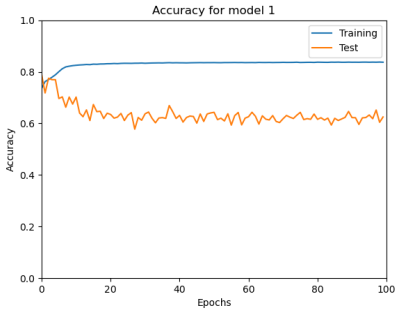
(a) Accuracy for the cross-validation.



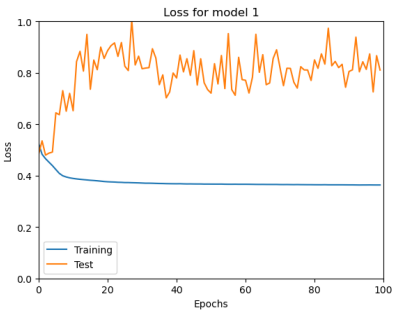
(b) Loss for the cross-validation.

Figure 4: Results for the cross-validation using our first model.

Finally we see how our model performs on the test set. In figure 5a we see that there is a considerable difference between the accuracy on the training set and the test set. In figure 5b we see that the loss follows a similar pattern. The deviation of accuracy and loss is far greater than the error we found from the validation sets. In table 2 we see that the accuracy on the test set is 0.624, as opposed to an accuracy of 0.837 on the training set. The loss on the test set was found to be 0.810, while the loss for the training set was 0.363. This is a far larger error than what we found from the cross-validation.



(a) Accuracy for the test set.



(b) Loss for the test set.

Figure 5: Results for the the test set.

Table 2: Results from running our model on the test set with 100 epochs.

	Accuracy	**Loss**
Training set	0.837 ± 0.003	0.363 ± 0.004
Test set	0.624 ± 0.003	0.810 ± 0.004

This could be due to an error in calculating the labels for the test set, or simply that the test and training set deviates too much from each other. However, given that the labels of the test set is created using the training set as a basis it's not likely this is the case. It would however be interesting to see how the model would fare against other test sets created through alternative methods. The AMS method used in this project was given from the starting kit provided by the Higgs challenge.
Another explanation for our results could be that our model is not performing as well as appears to. If our model guessed "background" for every example it would still achieve an accuracy of 0.66. Measuring how well it performs using other metrics than accuracy and loss would be a next step. We could for example see how well it distinguishes true positives and true negatives, which would probably be a better choice than simply showing how many of the labels it guessed correctly.

5 Conclusion

In this project we have produced a supervised machine learning model that so far has limited success in differentiating between a signal and background event on a given test set, despite showing promising results on the training set. We have seen that the best result on the training set was achieved by choosing a model with few nodes and layers. The second choice of hyperparameters yilded a better accuracy but the loss diverged as the model progressed through epochs, suggesting this choice is prone to overfit. The last option for hyperparameters gave worse results for both accuracy and loss.
While the model performs well on the training set showing no signs of overfitting or underfitting, there is clearly an issue when running this model on the test set. Our model achieved an accuracy of 0.837 and a loss of 0.363 on the training set, while the test set gave an accuracy of 0.624 and a loss of 0.810. This is a much larger error than the mean error we found from cross-validation,

which gave us a mean error of 0.003 and 0.004 for accuracy and loss respectively.
The next step for this project would be to be to include other metrics than simply accuracy and loss. By only including these two metrics we get a limited view on how well the model actually performs. Further, it would be interesting to see how other types of machine learning algorithms performs. Having tested out these options, one could also investigate how impactful our choice in preprocessing is on the end result. Our choice of setting all undefined values to 0 after normalizing seemed like the most sensible choice, but other options could yield better results.

References

[1] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balàzs Kégl, and David Rousseau. The Higgs boson machine learning challenge. In Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau, editors, *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *Proceedings of Machine Learning Research*, pages 19–55, Montreal, Canada, 13 Dec 2015. PMLR.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 6. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 5. MIT Press, 2016. http://www.deeplearningbook.org.