Use of Machine Learning in Analysis of Particle Data

Paul A. Cordova*
Physics Department, University of Virginia, Charlottesville, VA 22904, USA
(Dated: May 11, 2018)

Fixed target scattering experiments create large quantities of data in their detectors. Parsing through and analyzing this data is key to making insights and learning new physics. Certain interactions between the target and beam create corresponding end products. Furthermore, different decay channels can lead to the same end products. This creates a classification problem: which channels did the event come from? By using particle physics symmetries and conservation laws, analytical techniques can be employed to extract signals out of the multichannel phasespace. The topology of interest can often be separated from a complex background by using some simple kinematic constraints but often these tools are limited. A multivariate analysis built on many discriminating variables can be used to improve the quality of the physics extracted from the scattering experiments data. Machine learning algorithms optimized for application in multivariate classification along with a high performance computing platform affords the opportunity for specialized technique comparison. In this paper, various machine learning algorithms are trained on theoretical labelling of the events and then used to classify which channel an event corresponds to. The machine learning methods are explored using ROOT's TMVA libraries as well as TensorFlow to see how well certain classification algorithms perform. Overall, TMVA's BDTB, BDTMitFisher, MLP, and DNN algorithms (decision trees and neural nets) seemed to perform the best at classification of the channel of interest. Furthermore, an open source SOM (self-organizing map) algorithm written with TensorFlow was used to test how unsupervised machine learning compares with the established classification algorithms in TMVA. Overall, the self-organizing map performed roughly on par with the TMVA algorithms tested. This paper explores the methodologies and results of the algorithms on simulated data.

* pac3nc@virginia.edu

I. INTRODUCTION

In nuclear and particle physics, scattering experiments produce large quantities of data during tracking. This is especially true in large acceptance detector systems used in hadron spectroscopy. These interactions are scattering experiments in which a beam hits a target and particles are produced by the collision. The particles produced by the scattering experiment are then measured in a detector and momentum, angular, mass, and timing data are collected. This data then is then used with particle identification to identify the final particles produced by the interaction and detected in the detector. These final particles are decays from larger particles (resonances, etc.) produced by the interaction. Each of these decay modes are channels, and channels have different distributions of observables associated with them. These distributions are predicted by theoretical physics, and thus we can use the observable data to figure out which channel the particle came from. However this is not a trivial task on real particle data. So, what is often done is cuts are made by the physicist on the data (backed by theory) in order to classify and extract the channel a particular event came from. Instead of using relying on theoretical principles and human decision making to classify an event, it is possible to automate the process by training a machine learning algorithm on theoretical Monte Carlo simulations and then have it automatically classify events based on training. This report explores various ways in which this machine learning can be used to extract these channels to analyze the particle data.

The goal of this report is to explain the physics behind the two decay channels of interest from the interaction, explain how the Monte Carlo generates the data, and ultimately explore the efficacy of the various machine learning algorithms in classification of the decay channels. The primary goal of this report is to focus on which machine learning tools work best to classify the data and to explore methods of interfacing new toolkits (such as TensorFlow) with particle data.

II. THE INTERACTION

A.

The signal:

$$\begin{aligned} \gamma p &\longrightarrow K^+\Sigma^{0*} \\ \Sigma^{0*} &\longrightarrow \Lambda\pi^0 \\ \pi^0 &\longrightarrow \gamma\gamma \\ \Lambda &\longrightarrow p\pi^- \end{aligned} \tag{1}$$

The background:

$$\begin{aligned} \gamma p &\longrightarrow K^{+*}\Lambda \\ K^{+*} &\longrightarrow K^+\pi^0 \\ \pi^0 &\longrightarrow \gamma\gamma \\ \Lambda &\longrightarrow p\pi^- \end{aligned} \tag{2}$$

These are the two channels to be investigated using machine learning. It is important to note here that the end products are identical (π^0, p, π^-, and K^+). Hence if we make a cut on the particle identification on real data, we don't know which channel a given set of variables came from! Now in real world data, there are many more than two channels that lead to our end products. However some decay channels are more likely than others given the end products. These two channels have radically different event data so they are an easier problem for machine learning techniques to classify than other channels. However, ideally we hope machine learning techniques could improve into the future to allow analysis on more difficult comparisons, such as when in Eq. 2 the K^{+*} decays into a $K^+\gamma$ instead of $K^+\pi^0$. This is harder to train on because the momentum of the γ and π^0 are very hard to differentiate. In addition, in real world data, the γ channel occurs two to three orders of magnitude less often than the π^0 channel.[1]

These channels are simulated using GSIM, which simulates the CLAS detector at Jefferson Lab using GEANT. The CLAS detector is a spectrometer used to detect particles produced by interactions of a photon beam with a cryogenically cooled target. For the particular channels we are interested in, the target is a proton. The detector uses scintillator and drift chambers to detect and make measurements on charged particle data. From there, conservation of energy, momentum, etc. can be used to calculate the momenta of the chargeless particles and time of flight cuts are used to create discrete data points of each event. [8]

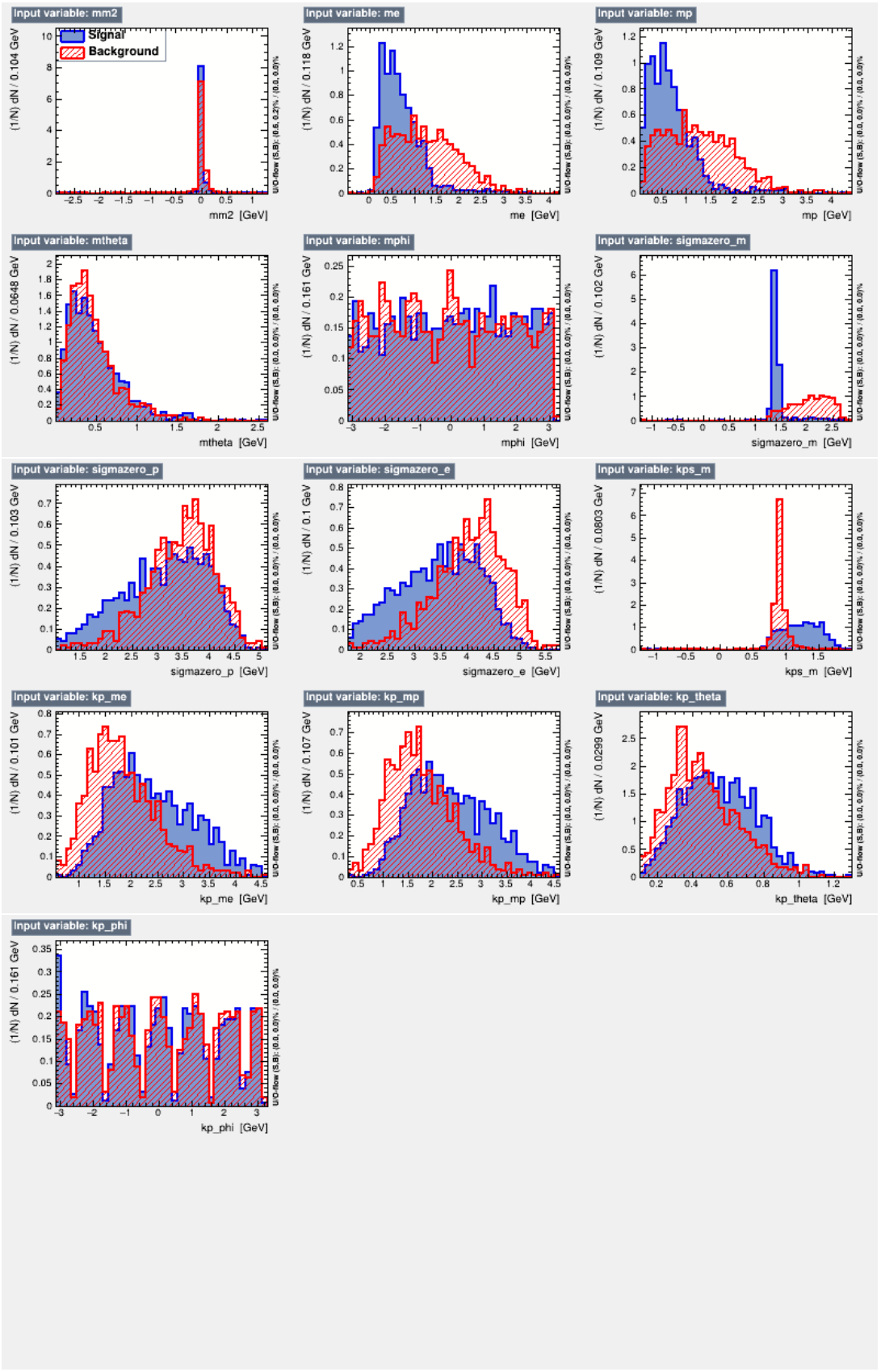
The variables that are used from the interaction and fed into the classification algorithm are the missing mass squared (which should be the mass squared of the π^0), the missing energy, missing momentum, Σ^{0*} mass, K^{+*} mass, K^+ energy, K^+ momentum, and K^+ angle ϕ. The observables that the detector gathers are the four-vectors of the K^+, p, and π^-. From those four-vectors, the vector sum of certain combinations of the four-vectors will result in the four-vectors of the parent particle from Eq. 1 and Eq. 2. This is purely a consequence of conservation of momentum. For example, the four-vector of the Λ is just the vector sum of the four-vector of the p and the π^-. These four-vectors just follow the standard formulas for special relativity. Namely, the four-vector is just $\mathbf{p} = (\frac{E}{c}, p_x, p_y, p_z)$ and the mass can just be calculated by Eq. 3.

$$\frac{-E^2}{c^2} + |\mathbf{p}|^2 = -m^2c^2 \tag{3}$$

For charge-less particles, we just calculate the missing four-vector by again using conservation of momentum. The detector only detects three particles as end products, furthermore the vector sum of these the particle's corresponding four-vectors does not equal that of the incoming four-vector. Therefore so long as we believe in the law of momentum conservation, the missing four-vector is just the incoming four-vector minus the vector sum of the detected four-vectors. Therefore, each data point has associated with it each of the variables that we are training the algorithm with. This is true regardless of the decay channel the end products originated from. For example, each data point has associated with it a Σ^{0*} four-vector, even if it that isn't the channel where the particles came from. This four-vector is just the vector sum of the Λ and the missing four-vector. The distribution of observables (energy, mass, momentum) might be different depending on which decay channel the particles came from, but each channel does have these vector sums. If the data points did truly come from the Σ^{0*} channel, then the distribution of data points would show a sharp peak for the mass of the Σ^{0*}. Whereas if the particles came from the other channel, then the distribution wouldn't have a sharp peak but a distribution. Fig. 1 shows a graph of the variables and their distributions in the signal and background channels. Kinematic fitting variables can also be used to improve accuracy, although they weren't used here as they weren't producing very good results and were actually disrupting the maps and classification algorithms. With some tuning though, they should be useful.

There are many channels of interest in particle physics, however this channel is useful for testing machine learning techniques as it is an "easy" channel to classify. In principle a robust and efficient machine learning algorithm should be useful for other more difficult classifications where the event data isn't as easy to find patterns for.

Figure 1: Distributions of each of the kinematic variables in the signal and background channels.



III. TMVA ROOT

A. Generating the Data

To generate the data and get it into a format where it is view-able in ROOT, we use the `genr8` generator from the Jefferson Lab CLAS libraries. The generator reads an input file which consists of the channel we are interested in. A different input file is needed for each channel that you want data for. Then this is converted into BOS files and are ran through GSIM libraries, which tracks the particles through a simulation of the CLAS detector. The BOS files are then read into our own programs which read BOS files and convert them into ROOT files. Once the data is in a ROOT file, we make certain cuts on the data based on whether it has the particles we want, if it obeys conservation of energy and momentum, etc. Then we put the variables we want into a final ROOT tree. Once the variables of the channels are in corresponding ROOT trees, the TMVA libraries read the trees (note, two trees are needed: one for signal and one for background). From there were can use the built in algorithms to train on the data and see how well they did, picking out the best algorithms. The four algorithms that performed the best were the deep neural network (DNN), multilayer perceptron (MLP), and the boosted decision tree (BDTB and BDTMitFisher).

It is important to note that each of these algorithms ran on the data involve "supervised" machine learning. This means that during training of the algorithm, the classification algorithm knows the correct result of the classification. It then optimizes to create a set of neurons, decision tree weights, etc. so that it minimizes the rate of incorrect classification. This is in contrast to "unsupervised" machine learning which is explored in the TensorFlow part of this paper. Unsupervised machine learning involves an algorithm which does not know the correct classification. Instead training just involves trying to exploit regularities in the data so as to create distinct clusters of similar data points. Unsupervised learning has applications beyond classification, but it can be used to classify as similar data points will cluster around similar neurons. However because the correct classification isn't given to the algorithm, the clusters don't "mean" anything. You can also have greater or fewer clusters than classifications, depending on how well the unsupervised learning algorithm is doing compared to what you want.

B. Neural Networks

Both the MLP and the DNN are essentially just variations of a neural net. Each have certain modifications in order to improve the accuracy of the response. The method by which neural nets work is by using hidden layers of neurons with associated weights to attempt to minimize a cost function via gradient descent. The cost function is essentially just whether or not the neural net makes the correct prediction for that data point (all the variables) in training. If it makes an incorrect classification, then cost is higher. The weights of the neurons in the various layers are adjusted until the neural net is making classifications that are consistent with the training data. The trained neural net can then be applied to classify new data.[4]

C. Decision Trees

Fig. 2 shows an example of how a decision tree operates. The weights of the variables and where to cut (sex, age, sibsp) are optimized on the training data so as to create the most accurate classifications.

Whereas in a neural network, the hidden layers and weights of neurons are seen as a "black box" where the weights don't really have much meaning, a decision tree simply makes cuts on the values of the variables. This makes it easier to interpret what the tree is doing. It attempts to optimize the accuracy of the classification by selecting ideal thresholds by which to cut. These thresholds, could in principle be viewed as a tree to the user to see if what the decision tree is doing makes sense. Both the BDTB and BDTMitFisher algorithms are decision trees. The "boosted" part of a decision tree is just a technique to improve the efficiency of the decision tree. Furthermore the differences between these two different maps are just minor tweaks and adjustments to the technique to optimize and possibly perform better.[5]

D. Results

Fig. 3, Fig 4., Fig. 5, and Fig. 6 show the response for each of the highest performing algorithms. The way to interpret these plots is to see how well the reponse curves on the left are split between signal and background. From here the physicist/data scientist can cut the results depending on the purity required and the number of statistics

Figure 2: Example decision tree for survival probability on the Titanic given some variables.[5]

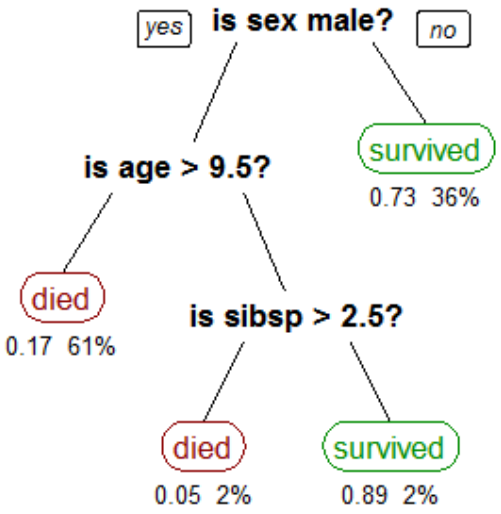
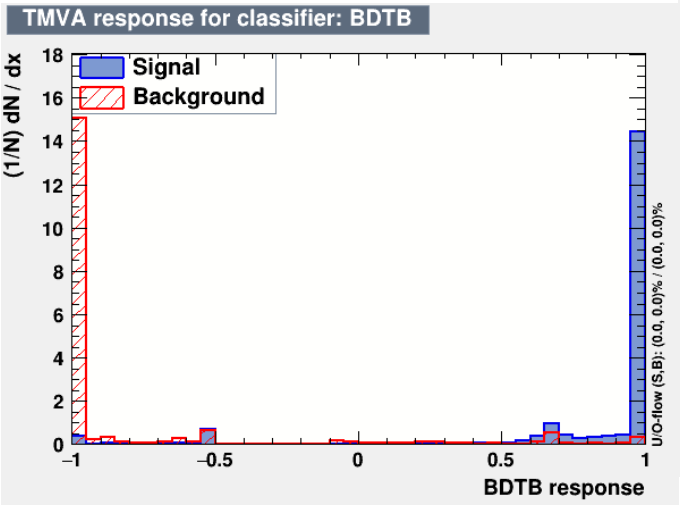
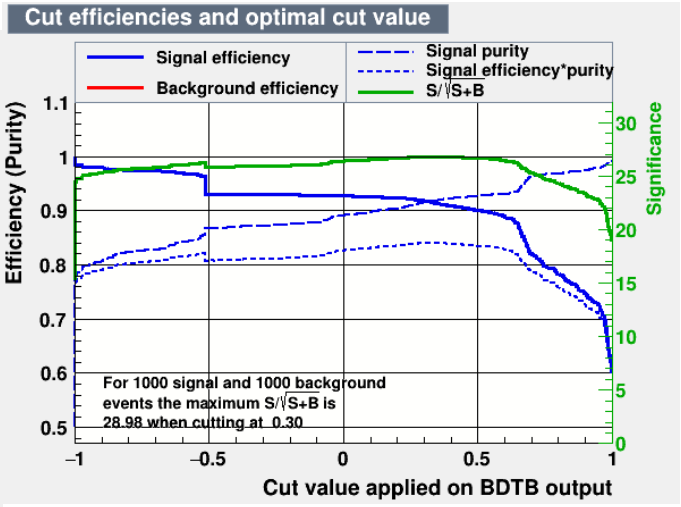


Figure 3: BDTB classification results.



(a) Graph of the number of events of each signal and background given response of the algorithm.

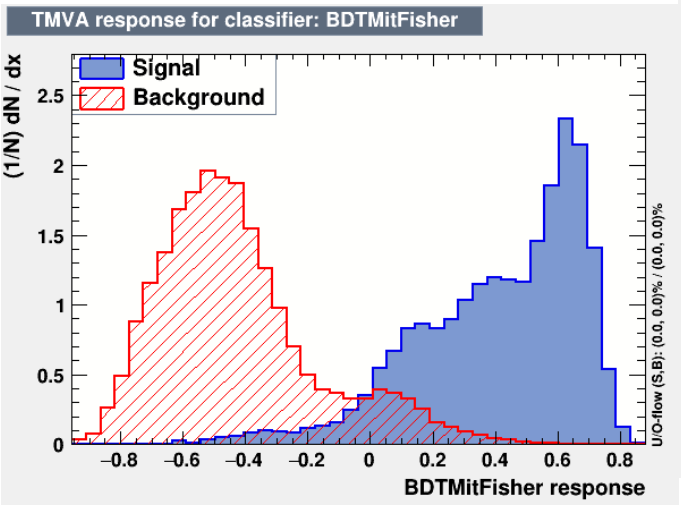


(b) Graphs of purity, signal efficiency, etc. of the various locations to cut from the classifier. This can assist with decision making of where put classification thresholds given how much purity and statistics we want.

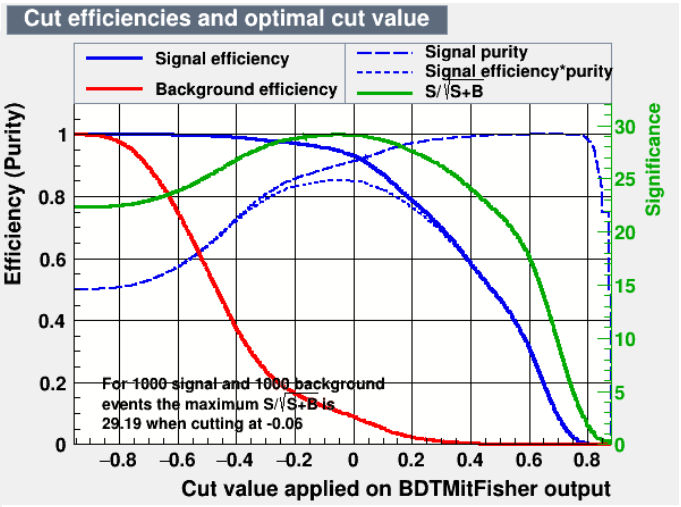
required. There is usually a trade-off (higher purity means fewer statistics) and for a better algorithm there is less of a trade-off. The right graphs show the various metrics used. Frequently a cut on the maximum of $S/\sqrt{S+B}$ is used to get a healthy balance between purity and statistics. However for higher purity applications, it may not be ideal to cut based on this metric. In short, the better an algorithm is, the more pure each of the ends of the response are, and there is more separation between signal and background on the graphs to the left.

TMVA ROOT has many more algorithms, but these were the algorithms that performed the best. For this "easy" channel, the machine learning seemed to do quite well and is capable of classifying signal and background using the response from the algorithm. From these machine learning techniques, the physicist can now cut the events given a threshold of the algorithm response. For example, one might cut on an MLP response above 0.9, yielding a result with

Figure 4: BDTMitFisher classification results.

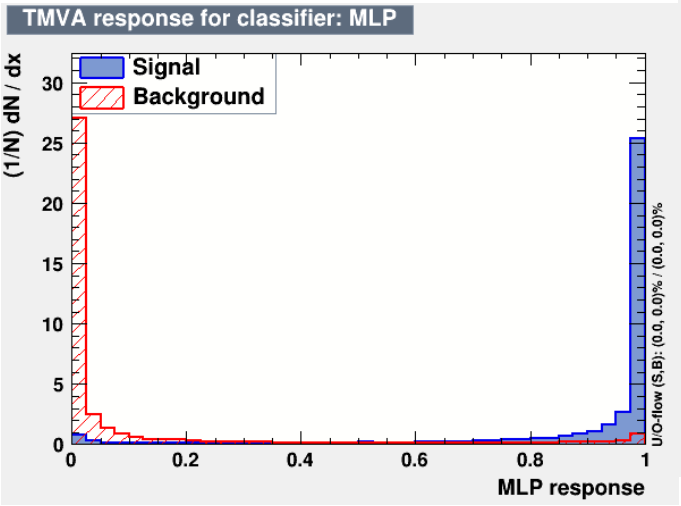


(a) Graph of the number of events of each signal and background given response of the algorithm.

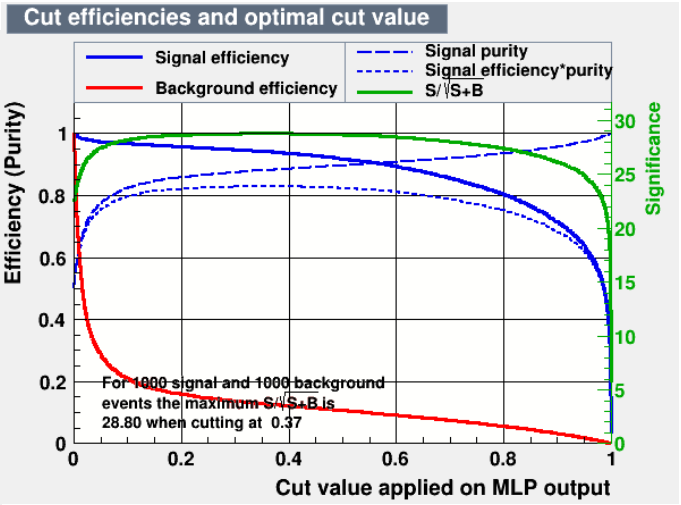


(b) Graphs of purity, signal efficiency, etc. of the various locations to cut from the classifier. This can assist with decision making of where put classification thresholds given how much purity and statistics we want.

Figure 5: MLP classification results.



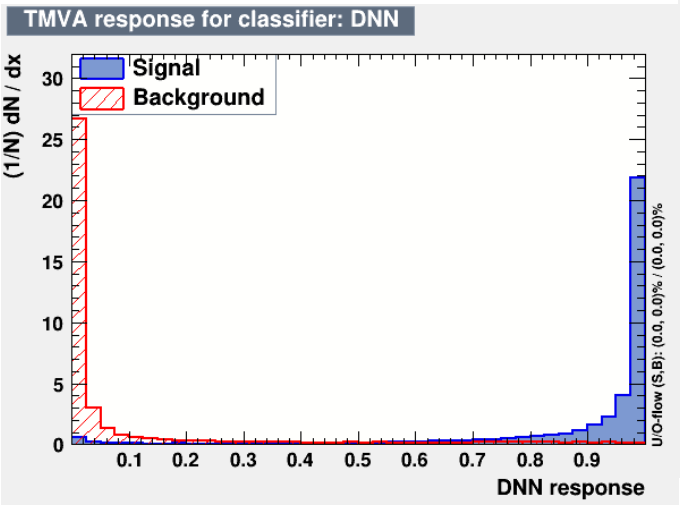
(a) Graph of the number of events of each signal and background given response of the algorithm.



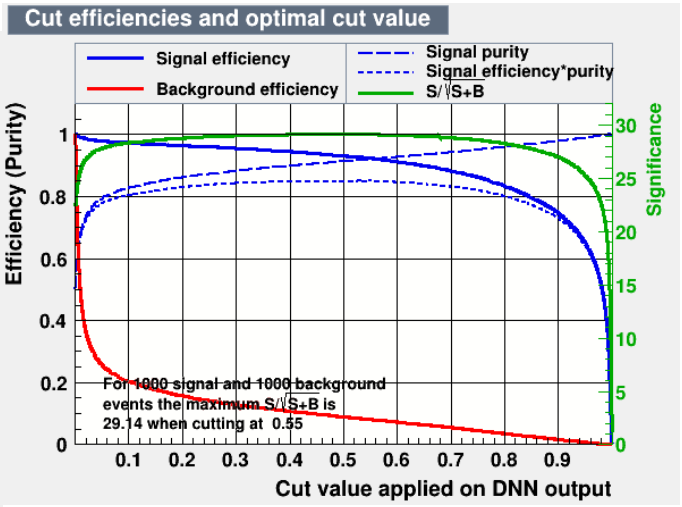
(b) Graphs of purity, signal efficiency, etc. of the various locations to cut from the classifier. This can assist with decision making of where put classification thresholds given how much purity and statistics we want.

very high signal. Generally speaking, the physicist must make a decision on whether to cut for statistics or for purity. The higher the purity required, the worse the statistics (as data points are effectively thrown away). Many physicists like to cut where $S/\sqrt{S+B}$ is maximum. This sort of cut still maintains high purity for BDTB, MLP, and DNN algorithms but introduces less purity for the BDTMitFisher algorithm. It is also important to note that for another channel that was tried (the π^0 decay versus the γ decay for the K^{+*} channel) the response from the algorithms is not nearly as good at classification. Classification can be improved by careful selection of variables that we give the algorithm so that it best learns to classify. For example, if the kinematic fitting variables were organized properly, they may assist with classification of the more complex channel.

Figure 6: DNN classification results.



(a) Graph of the number of events of each signal and background given response of the algorithm.



(b) Graphs of purity, signal efficiency, etc. of the various locations to cut from the classifier. This can assist with decision making of where put classification thresholds given how much purity and statistics we want.

IV. TENSORFLOW SELF-ORGANIZING MAP

Figure 7: Example self-organizing map for voting patterns in Congress.[6]

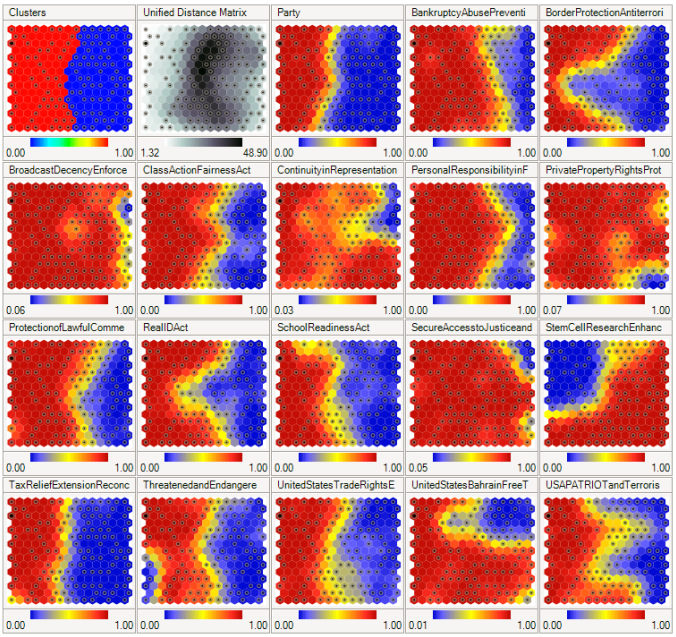


Fig. 7 shows an example self-organizing map based on voting patterns in Congress. It shows clear clustering on how politicians vote on party lines, and it also shows deviances from party lines as well as which issues have bipartisan support. This mapping comes from natural sorting of the data due to regularities in the way that politicians vote.

A self-organizing map (SOM) is a method of unsupervised learning. Whereas the boosted decision tree and neural nets from the TMVA library in ROOT train off the labeled data, a self-organizing map does not require the data be labelled in a particular scheme. In principle a self-organizing map will be able to cluster data simply by finding regularities in the data. Furthermore a self-organizing map has the capacity of "dimensionality reduction" by which

it reduces our 8-dimensional problem into a 2-dimensional problem, although this feature isn't used very much in our use case.

The manner by which a self-organizing map works is fairly straightforward. First a $M \times N$ matrix of "neurons" or "nodes" is created with random weights associated with them. Each neuron has an n dimensional "weight" vector associated with it. n is the total number of variables associated with the data. For example: one might want to train the SOM on the missing mass squared of the interaction, the mass of the K^{+*}, and the mass of the Σ^{0*}. If we were just training off of these variables, then $n = 3$. M and N are arbitrary and select-able by the user, however many data scientists suggest as a rule of thumb that the total number of neurons $M \times N$ should be equal to $5\sqrt{P}$ for P total data points that we are training on. It is also possible to have $P \ll M \times N$ or $P \gg M \times N$ to achieve different results. For our use case, we will approximately follow the rule of thumb. Once these neurons are set up, random values are filled into the weights of the neurons, and training begins. The neurons begin to "compete" for the data in the n dimensional space and move towards the data points. The training is done such that nearby neighboring neurons will gravitate to similar data points. Neuron groups will eventually cluster towards different distinct areas of the data, allowing classification of these distinct groups of data. Each region of data that clusters near neurons has some sort of similar feature associated with it (the SOM does not tell us what that feature is though). The neurons are trained for many iterations (in our case, 400 iterations) to reach and form clusters on the data, each iteration moves the neurons closer to the the final clusters.[7]

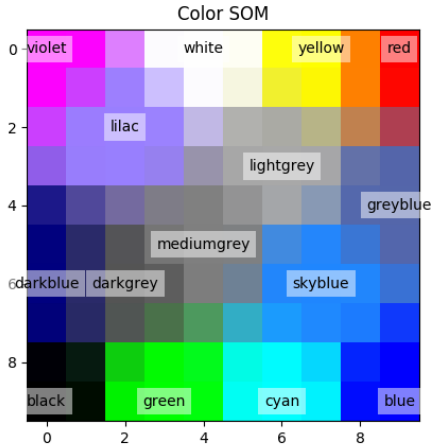
Since the self-organizing map is unsupervised and has no idea what the underlying structure of the data is, the clusters don't necessarily mean anything. However we (the human) interpret, and hope, that each cluster will correspond to the signal and background of our interaction.

A. Advantages and Disadvantages of TensorFlow

So how does TensorFlow compare with ROOT's TMVA? There are many advantages and disadvantages to TensorFlow. The main advantage is that we are able to use completely new algorithms that aren't in the TMVA libraries. The self-organizing map algorithm, for example, is not present in ROOT's libraries. TensorFlow gives us the ability to deploy new machine learning algorithms and try them. The downside is you have to roll your own algorithms. Self-organizing maps are not a pre-built algorithm in TensorFlow, unlike how MLP, DNN, BTDB, and BTDMitFisher are built into TMVA. This makes deployment of the algorithms much more difficult. Fortunately a thriving and active open source community means we are able to find implementations of algorithms such as SOM's and use them. For our use case, we used Sachin Joglekar's open source implimentation (modified as some of the functions were deprecated).[2]

The code by default works with neurons classifying and generating neurons with similar color to the input data points, where the input data points are just fifteen pre-defined colors. This can be seen in Fig. 9.

Figure 8: The default setup of the self-organizing map code. Neurons adjust to colors similar to the input data colors.



Another important obstacle to immediately running TensorFlow on ROOT data is that we have to get the library to interface with ROOT files. To do this, we follow a simple workflow: we convert the data in the ROOT file into

a `numpy` array using a Python library called `root_numpy`. From here we can use all sorts of Python data analysis libraries, including `numpy`, `pandas`, `scipy`, `matplotlib`, and of course `tensorflow`. [3] Conversion from a ROOT to a `numpy` array is fairly straightforward, although there are some quirks. However conversion from a `numpy` array to ROOT file has much more boilerplate code and is a larger hassle. In general, loading a ROOT file into Python isn't a super elegant and simple process, but it is doable without too much difficulty. It is also important to note that it isn't necessary to convert back to ROOT once we're done in Python. Although for our use case, it is sometimes preferable that we convert back.

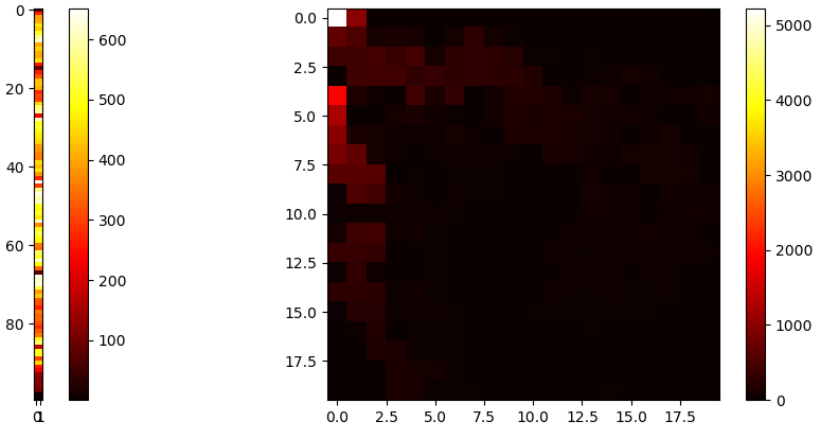
B. Results

Because the problem we are facing is a binary classification problem, we may want to force the data into two classifications, perhaps using only a 2×1 neural map. The idea is maybe one neuron would congregate and classify half the data as signal and the other would congregate and classify the other half as background. The problem was, for large statistics, one neuron won a vast majority of the data in the competitive training process, while the other won basically none of the data points. For smaller data sets there was some classification power with $70 - 80\%$ signal and $20 - 30\%$ background in one of the neruons, and vice-versa in the other. Which is decent, but it isn't really as flexible or as efficient in the cuts as ROOT's TMVA.

We found that this wasn't very effective at binary classification, so we tried a higher resolution 1-dimensional map of 10×1 and 100×1 to see if we would see two distinct clusters at the ends instead. This turned out to be better when we make cuts, but is still not ideal.

Fig. 8 shows a graph of both the 1-dimensional clustering and the 37-variable map. Neither were ideal for the classification application. The map of the 100×1 neural space shows that the hypothesis that clusters would be at the end points and represent signal and background is not true. Furthermore, it shows why a one dimensional map would be difficult to make cuts on, as there is topologically no real "peaks" or "valleys."

Figure 9: Self-organizing map for a 20×20 array of neurons.



(a) Failed attempt at using the SOM involving a one dimensional neural map of dimensions 100×1, clustering isn't as useful and is more difficult to cut on.

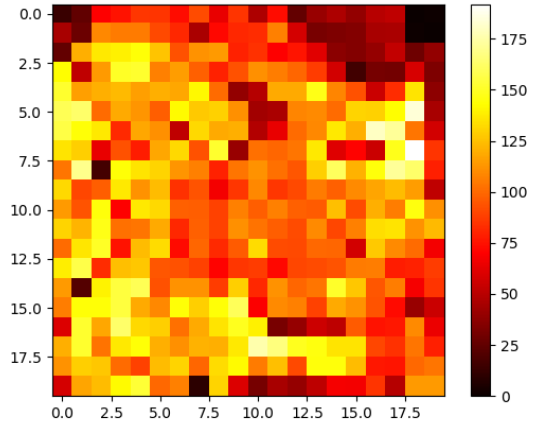
(b) Failed attempt at using the SOM on 37 variables. A majority of the data clustered to the top left neuron with mixed signal and background events.

Hence, even though this is a binary classification model, it still makes sense to use a large 2-dimensional map and look for clusters there. This improves the versatility and level of control of the cuts. The problem is the interpretation of these clusters becomes significantly more complicated for the scientist to understand and cut on. Without a systematic approach, it is unclear how to cut from the raw map of clustering.

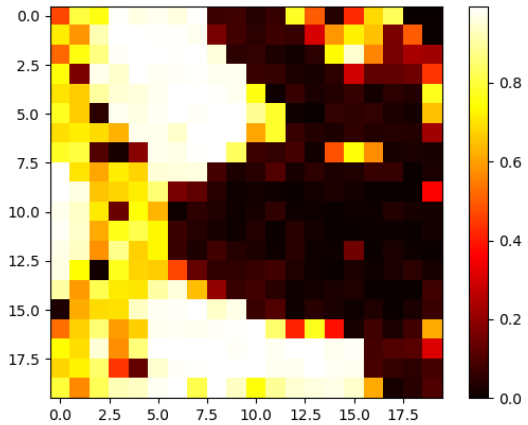
Fig. 10 shows the clustering in a 20×20 dimensional self-organizing map. On the left is the number of events associated with each neuron. Clustering of neurons is characterized by groups of high counts (clusters) separated by large regions of groups of low counts (valleys). This raw data on just number of events associated with each neuron just shows that the self-organizing map is organizing in some manner due to the regularities in the data, and is able to distinguish between certain types of data. Although because the algorithm is untrained, it has no awareness of

whether or not a data point is signal or background. On the right, purity of signal events to background events is shown. This proves that although the self-organizing map has no idea what it is categorizing, or into how many groups, it does manage to separate the two types that were specified by the simulation. This is evident by the fact that the clusters of nodes are associated with either pure signal or pure background with very few neurons between the separation with mixed signal and background.

Figure 10: Self-organizing map for a 20×20 array of neurons.



(a) Self-organizing map of the number of events associated with each neuron. Non-uniformity shows clusters. Color of heatmap is number of events.



(b) Self-organizing map of the location of the signal events. Clear separation indicates the self-organizing map is successfully splitting up the signal and background data. Color of heatmap is the level of signal in the neuron (e.g. 0.9 means 90% signal).

The idea is that the user would view each of the clusters and determine what they are by creating a separate data set of the data points in each of the clusters. The problem is, without some kind of metric to know whether or not a point is signal or background, our cuts aren't really useful for aggregating and sorting the data. Furthermore, ideally classification of just two clusters is preferable as the user can impose meaning on the clusters (one being signal, the other background). However, there are three clusters in the figure! Without some indication of what is what, it is very difficult to make a binary classification from the map. Perhaps one could argue that the two clusters on the left of the figure are one classification and the one on the right is another given the "valley" in between the peak regions. However the process involves a lot of trial and error without a systematic way to cut the data, it isn't particularly efficient.

Furthermore, the self-organizing map was also ran on additional variables (including kinematic fitting variables, and other kinematic variables) for a total of thirty-seven variables instead of the usual eight. As seen in Fig. 8 next to the 1-dimensional map, the map of 37-variables clearly shows that adding in variables that are not useful really hurts the clustering ability of the SOM.

To remedy the problem of difficulty in cutting the data, the "unsupervised" learning and clustering in the map can be converted to a "supervised" learning fairly easily. All that is needed is for a spectator "signal" variable with the label to be passed with the data but not used in training, and then cuts can be made with this signal variable. Regions of neurons with data points associated with this signal variable can be classified as a signal neuron, while regions of neurons not associated with the signal variable can be classified as background. Then cuts can be made on the regions trained.

Fig. 11 shows the comparison of the raw unsupervised maps as well as the distribution of signal in the map for both the data set the neurons were trained on in addition to a completely new data set with the same channels in it. The similarity of the maps shows that the trained map is not over-trained and can be deployed over data with similar channels.

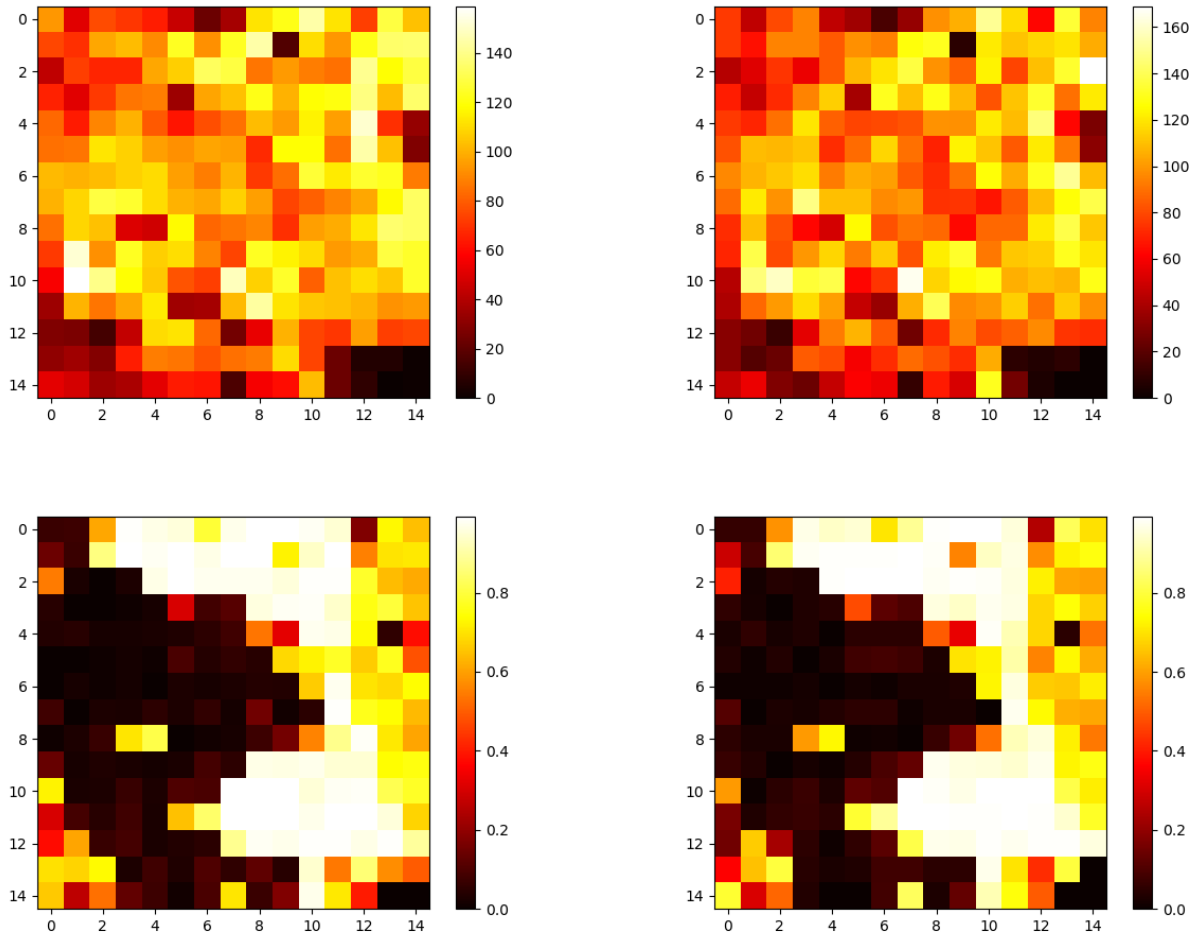
Fig. 12 shows the signal and background regions in the map, again using the labelling variable to plot where the events lie. The map was not trained on this variable, it is just used to determine if the map is actually separating the two different classifications.

Since signal and background are clearly separated in this map, it stands to reason that good cuts could be made

Figure 11: A comparison of neural training maps from the training data set and a separate similar data set.

(a) The clustering of the unsupervised SOM for the training data set, as well as the distribution of the signal with the supervision label.

(b) Clustering of the unsupervised SOM for a different data set of the same channels, as well as the distribution of the signal with the previously trained neurons.

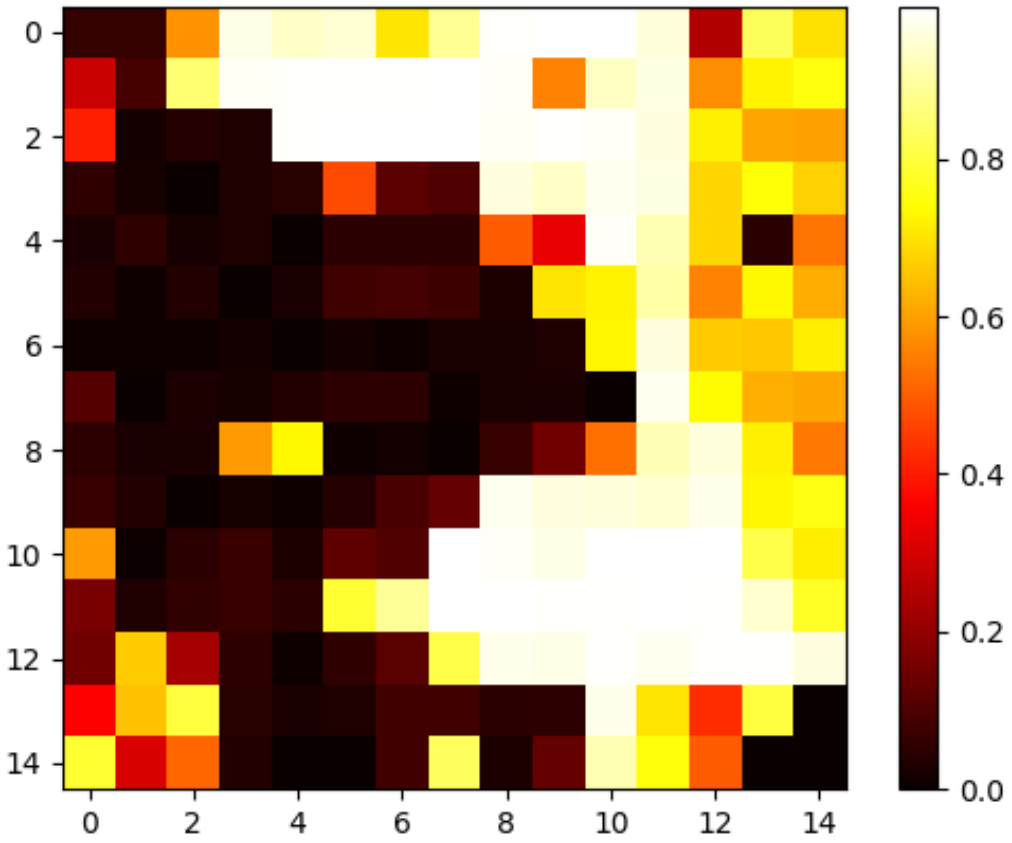


with these regions. However to avoid risks of viewing an over trained map, it is ideal that the regions are specified and the neurons are trained on one data set, and then the statistics and neurons are applied to a completely different data set characterized by the same channels. This allows for the capacity to make a higher resolution cut on the neurons, and achieve higher levels of purity if required.

Fig. 13 shows plots similar to what is given in the ROOT TMVA results. The response of the algorithm between the two points allows for comparison of purity versus number of statistics in a similar manner to the TMVA output. Similar to some of the other algorithms, each of the ends involve very high purity of statistics whereas data points with responses in the middle are associated with lower purity and more mixing. To maximize the $S/\sqrt{S+B}$ metric, it is recommended to cut at a response of 0.382353, although this will likely sacrifice a lot of purity due to the high amount of statistics and mixing in the range 0.6 to 0.8. Hence, if the physicist/data scientist prefers higher purity, one might cut closer to the ends and throw the data points out.

Overall, by using the signal variable to determine the region to be cut, the results are around as good as the other algorithms in ROOT, which makes it a viable algorithm if the methodology is tweaked. This shows a very pleasant benefit of using self-organizing maps compared with the algorithms in ROOT, which is the versatility and level of user choice. One could come up with a way of cutting data without using the labelling variable. Furthermore there may be better ways of cutting the clustered neurons off of say a different variable or the counts themselves. The downside is that, the user needs to know what they are doing and what is going on under the hood. In addition

Figure 12: Self-organizing map for a 15×15 array of neurons, classified by the signal training variable. Map was trained on a different dataset for the same channels. Color of heatmap is the level of signal in the neuron (e.g. 0.9 means 90% signal).



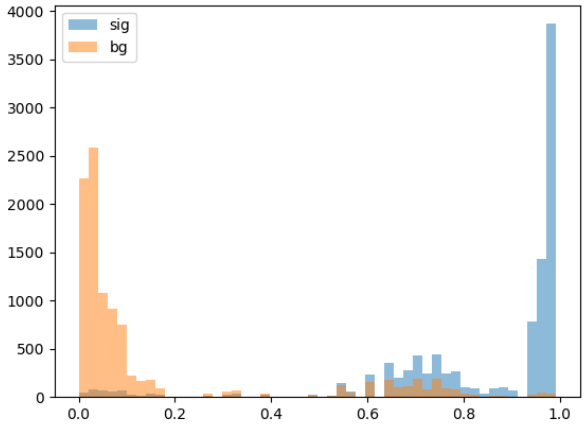
there are infinitely many ways to approach the clustered data from the user end, there is no "optimal" way due to the unsupervised nature of the map. This means a clever data scientist is needed to make decisions on how to cut and approach the map.

Furthermore, if techniques are improved to truly utilize the "unsupervised" part of the algorithm instead of relying on the training signal variable, then it may even be possible to extract new theory from self organizing maps as it finds new patterns in the data to form clusters that standard analysis would be unable to find.

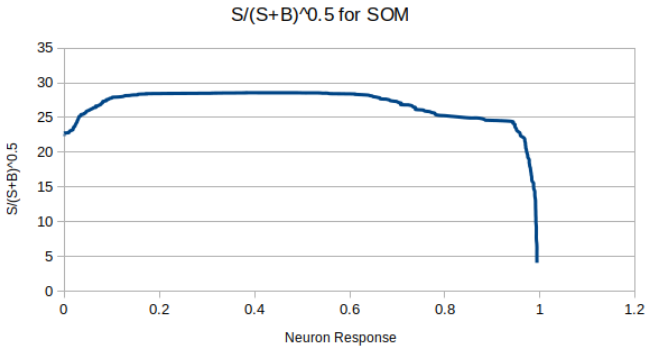
In conclusion, the performance of self-organizing maps compares with the performance of the established algorithms in ROOT. It simplifies the complex problem of classification on many variables into a neat abstract clustering in a two-dimensional space and allows us to potentially learn new relationships between variables if used without supervision. However, for our use case, it seems to make more sense to use SOM's in a supervised manner and categorize neurons based on a signal labelling scheme. Otherwise it is very difficult to look at the unsupervised map and conclude what each cluster actually is. SOM's are also less black-boxy in the sense that we see and can interpret the weights of the neurons, whereas in a neural network the weights are very difficult to interpret. Although, SOM's are still more of a black box compared with a decision tree.

Figure 13: The reponse of the classifier with the number of signal and background events, allowing the user to cut based on response, as well as the graph of the efficiency of various cut points.

(a) Response for classifier. Higher ranked neurons are on the right, lower ranked neurons are on the left. Bins are measured by number of statistics.



(b) Efficiency ($S/\sqrt{S+B}$) of cutting on the neurons ranked on by highest signal from training data. Maximum at response = 0.382353. Normalized to 1000 events.



V. CONCLUSION

In conclusion, machine learning techniques in both ROOT TMVA (decision trees and neural nets) and TensorFlow (self-organizing map) proved to work for classifying and differentiating these two channels. They may not work as well for more difficult channels with similar data that is hard to differentiate. The self-organizing map performed reasonably well when a training variable was used to identify neural regions with high signal. This means the map did differentiate between signal and background and was roughly on par with the TMVA algorithms. TensorFlow offers an excellent framework to test new algorithms that aren't built into TMVA, allowing more work and research to be done to improve classification capabilities. However the workflow is not as closely integrated and hence it isn't necessarily the best tool to use for implementing machine learning on real data. Instead, it is ideal for testing new algorithms that could later be re-implemented into ROOT once they've been established.

Overall, the most interesting result of this analysis is the efficacy of the self-organizing map. Not only is there some clustering apparent on the unsupervised result, but the supervised result (cutting on neurons based on neurons with high signal response) performed remarkably well compared with the supervised algorithms built into ROOT. Future research could involve figuring out clever way to cut the self-organizing map off of a variable that doesn't rely on cutting the map based on labels. Or to differentiate between signal and background clusters by using some kind of cut that separates two sides of a valley where there is no clustering. Furthermore, the techniques used to interface with ROOT with Python and TensorFlow could prove useful for testing out new algorithms and seeing how well they classify each of the decay channels.

VI. BIBLIOGRAPHY

[1] "The Review of Particle Physics (2017)" *Particle Data Group* http://pdg.lbl.gov/index.html
[2] Joglekar, Sachin. "TensorFlow-Examples", *GitHub* https://github.com/srjoglekar246/TensorFlow-Examples
[3] Dang, Aidan, et al. "Machine Learning with TensorFlow as an alternative to TMVA", *CERN* https://indico.cern.ch/event/505613/contributions/2228347/attachments/1346866/2045101/Oral_413_v3.pdf
[4] Sanderson, Grant. "Neural networks", *YouTube* https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
[5] "Decision tree learning" *Wikipedia*, https://en.wikipedia.org/wiki/Decision_tree_learning
[6] "Self-organizing map" *Wikipedia* https://en.wikipedia.org/wiki/Self-Organizing_map
[7] "SELF-ORGANIZING MAPS TUTORIAL" *ALGOBEANS* https://algobeans.com/2017/11/02/self-organizing-map/

[8] Williams, Mike. "Measurement of Differential Cross Sections and Spin Density matrix Elements along with a Partial Wave Analysis for $\gamma p \longrightarrow p\omega$ using CLAS at Jefferson Lab" Diss. Carnegie Mellon University, 2007.