Machine learning techniques for razor triggers

Marina Kolosova

University of Cyprus
marina.kolosova@cern.ch

Advisor: Jean-Roch Vlimant
jean-roch.vlimant@cern.ch

October 9, 2015

Abstract

During the CERN summer student programme 2015, I had the opportunity to work on a project related to the development of a neural network for triggering decisions. Following the research of the Caltech group at the CMS experiment, I developed a neural network which can predict if an event passes or not a razor trigger. Razor triggers are a part of the HLT menu and are used to distinguish the SUSY signals from the SM background.

I. Introduction

Supersymmetry (SUSY) is a hypothetical symmetry that postulates a partner force-carrying particle (boson) for each particle of matter (fermion) and vice versa. SUSY is one of the most compelling extensions of the Standard Model (SM) as it gives solutions to a number of yet unanswered questions, such as the origin of dark matter. Experimental searches for SUSY particles are focused on events containing energetic hadronic jets and leptons from the decays of pair-produced heavy particles, such as squarks and gluinos. These events are characterized by a significant missing transverse energy (E_T^{miss}) due to the two weakly interacting lightest superpartners (LSPs) produced in the decay chains.

The razor kinematic variables [1], M_R and R^2 are used to distinguish the SUSY signal from the SM background. They are motivated by events with pair production of squarks or gluinos where each of them decays to the LSP and a number of visible particles. By applying the megajets algorithm, each event is forced to have a dijet topology. The jets produced in the event are grouped into two megajets, containing at least one jet. The megajet four-momenta are defined as the sum of four-momenta of all the jets in each megajet. The jets are arranged in the dijet topology so that the sum of the invariant masses of the two megajets is minimum. Then, the four-momenta of the two jets are used to compute the razor variables, M_R and M_T^R, defined as:

$$M_R \equiv \sqrt{(|\overrightarrow{p}^{j_1}| + |\overrightarrow{p}^{j_2}|)^2 - (p_z^{j_1} + p_z^{j_2})^2} \tag{1}$$

$$M_T^R \equiv \sqrt{\frac{E_T^{miss}(p_T^{j_1} + p_T^{j_2}) - \overrightarrow{p}_T^{miss} \cdot (\overrightarrow{p}_T^{j_1} + \overrightarrow{p}_T^{j_2})}{2}} \tag{2}$$

where $\overrightarrow{p}_{j_i}$, $\overrightarrow{p}_T^{j_i}$ and $p_z^{j_i}$ are the momentum of the i^{th} jet, its transverse component and its its longitudinal component, respectively. E_T^{miss} is the magnitude of $\overrightarrow{p}_T^{miss}$. The razor variable M_T^R express the transverse momentum imbalance while M_R express the mass scale of a particle beyond the SM. The razor dimensionless ratio is defined as:

$$R \equiv \frac{M_T^R}{M_R} \tag{3}$$

II. Machine Learning techniques for Razor Triggers

Razor triggers are a part of the CMS high-level trigger menu. They require events with two or four particle-flow jets with a minimum transverse momentum of $p_T > 80\ GeV$ and $p_T > 40\ GeV$, respectively. Moreover, they require events to pass cuts on the razor variables M_R, R^2 and on the product $(M_R + 300) \times (R^2 + 0.25)$. With all this in mind, we would like to see if we could use machine-learning techniques for razor triggers. Therefore, the main purpose of my project is to build and train a neural network that responds to a razor trigger. The neural network should predict if a an event passes a razor cut, or not.

My project begins with the generation of synthetic data. The main reason why we use synthetic data is that we can generate and train the network on as many samples as we want, in contrast to Monte Carlo data where the number of samples is limited. However, one million samples were enough for our purpose. The events contain missing transverse energy and jets with a certain kinematic distribution. For each event, the components of missing transverse energy in x and y direction are given by a Gaussian distribution with a mean value equal to zero and a variance of three. The missing transverse energy is given by $MET = \sqrt{MET_x^2 + MET_y^2}$. The distributions of $MET_{x,y}$ and MET are shown in plots 1 and 2 respectively.

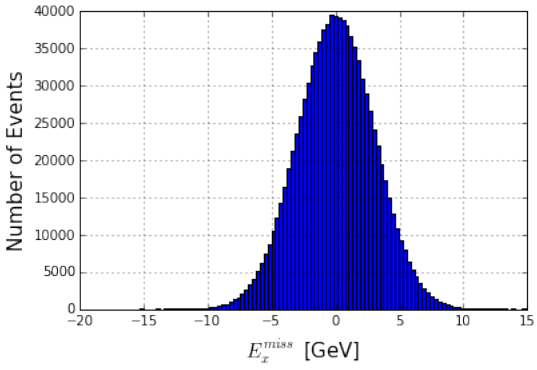

Figure 1: *$MET_{x,y}$ distribution*

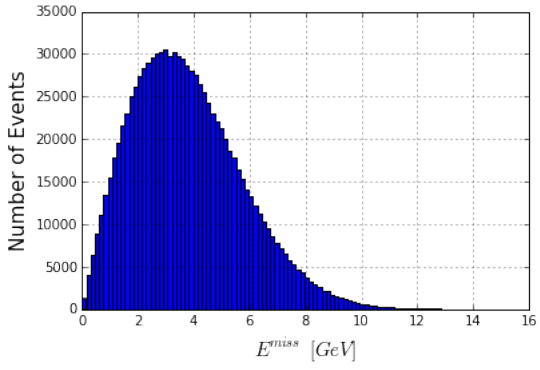

Figure 2: *MET distribution)*

The number of jets per event (plot 3) is given by a poisson distribution around six. For each jet, the mass (plot 4) is given by an exponential distribution with a lambda equal to 0.165 and the energy-momentum relation:

$$E^2 = (pc)^2 + (m_0c^2)^2 \tag{4}$$

defines the energy of the Jets.

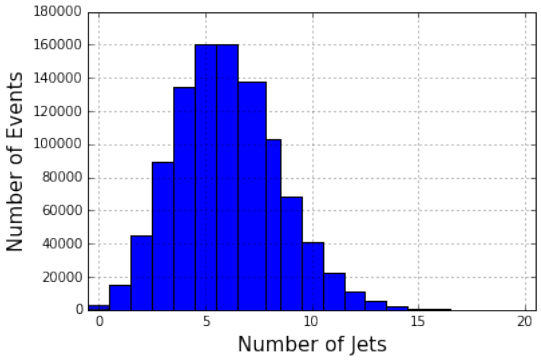

Figure 3: *Number of jets per event*

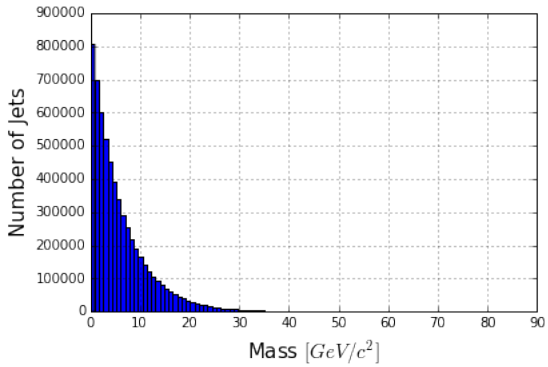

Figure 4: *Mass distribution)*

The components of momentum in x and y direction are given by a gaussian distribution with a mean value of zero and a variance of three while p_z has a variance of six.

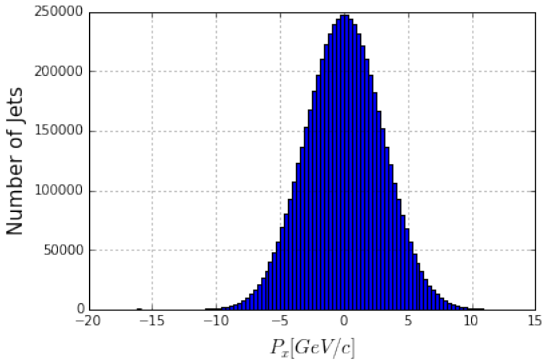

Figure 5: $p_{x,y}$ *distribution*

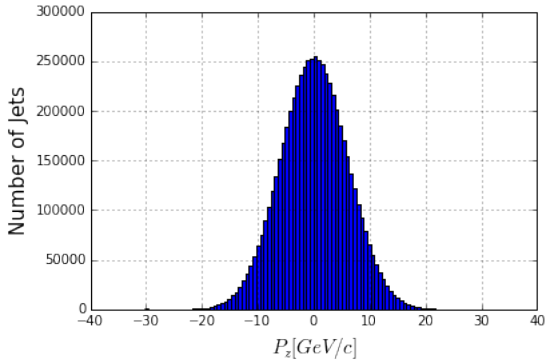

Figure 6: p_z *distribution*

After the generation of synthetic data, we applied the megajets algorithm in order to have a dijet topology for each event. The best arrangement of jets into two megajets is considered to be the one that minimizes the sum of the invariant masses of the two megajets. Then, having the megajet four momenta (the sum of four-momenta of the jets in each group) and MET, we calculated the razor kinematic variables, M_R, M_T^R and R. The razor variable R^2 as a function of M_R is shown in 7. As a simplified razor cut we require events to pass a cut on the product: $R^2 M_R > 0.6$. The percentage of events passing the cut is $\sim 60.8\%$, which give a Trigger Bit equal to 1 and the rest $\sim 39.2\%$ of events give a Trigger Bit equal to 0.

III. A Razor Neural Network

Computing the razor variables during the HLT requires some time. Neural networks could be faster for a trigger decision even at L1 trigger and could be implemented by using neuromorphic hardware.

For the development of the neural netrwork we used theanets package [2], which is a deep learning and neural network toolkit written in Python [3]. Theanets package supports GPU and is based on Theano package [4].

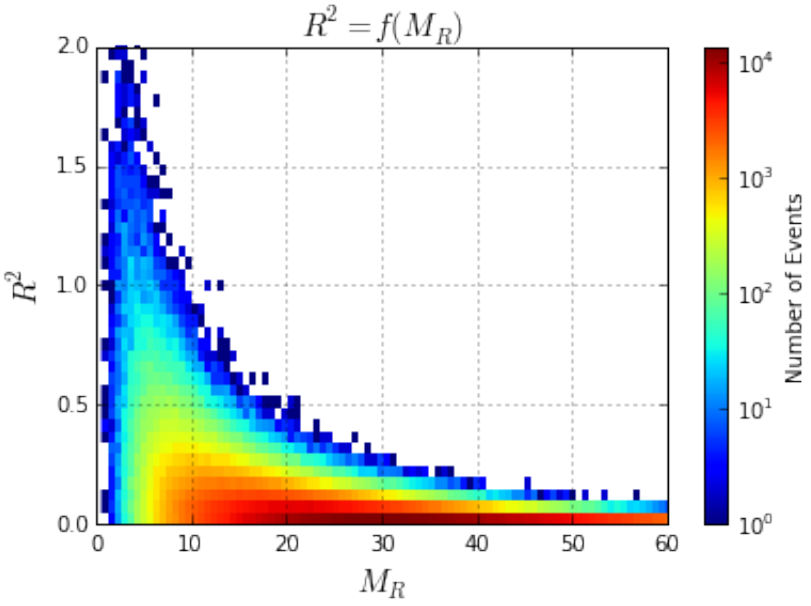

Figure 7: *Razor variables R^2 and M_R*

Since we want our neural network to predict the trigger bits, we use supervised learning to train it, which means that we have an input vector and a target for each event. Neural networks must have a fixed input size, so we set the maximum number of jets equal to 20. The network receives a vector with the four-momenta of all the jets in the event and the x and y components of missing transverse energy. Also, it receives the target (trigger bit), 0 for not passing the trigger cut, and 1 for passing it. The topology of the network is feed-forward (the information moves only forward, from the input nodes through the hidden nodes and then to the output nodes) with fully connected layers. The topology of the network is shown in figure 8. For the hidden and output layers I used the sigmoid activation function:

$$\Phi(S_i) = \frac{1}{1 + e^{-\alpha \cdot S_i}} \tag{5}$$

where S_i represents the total input of the i^{th} neuron which is equal to the sum over all individual inputs multiplied by their weights:

$$S_i = \sum_j \alpha_j w_{j,i} \tag{6}$$

We used 80% of the data set as a training set and 20% as a validation set, and tried to train on different number of samples, different number of hidden layers and neurons per hidden layers. At the plot 9 you can see how the network behaves as a function of the number of samples and hidden layers. The y axis is the figure of merit (FOM) which is defined as the number of events where the output of the neural network agrees with the trigger bit (target), divided by all events:

$$FOM = \frac{Number\ of\ events\ where: (TB = 1\ \&\ NN = 1) + (TB = 0\ \&\ NN = 0)}{All\ events} \tag{7}$$

The FOM increases with the number of samples but doesn't change a lot when we increase the number of layers. However, in order to improve the results, we have to tune all the training

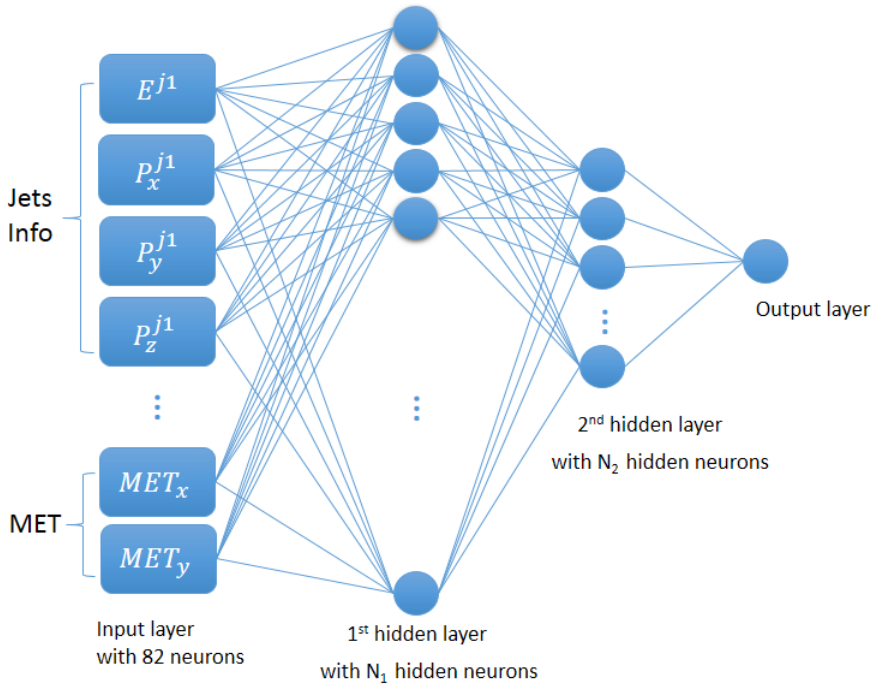

Figure 8: *The topology of the neural network*

parameters such as the learning rate, momentum, batch size, number of epochs and iterations, dropout, etc. You can find details for these parameters at [2].

- Learning Rate: The learning rate determines how much an updating step influences the current value of the weights:

$$w^{\tau+1} = w^{\tau} - \eta \nabla E(w^{\tau}) \tag{8}$$

where w are the weights, η is the learning rate and $E(w^{\tau})$ is the error function, which is defined as:

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \|y(x_n, w) - t_n\|^2 \tag{9}$$

where $\{x_n\}$, $n = 1, ..., N$ are the input vectors and $\{t_n\}$ the target vectors.

A typical value for the learning rate is in the range $[0, 1]$. A high learning rate makes the weights and the function that minimizes the error function diverge, which means no learning at all and a low learning rate makes the network learn very slowly. Learning rate can be combined with the next training parameter, momentum, to avoid local minima.

- Momentum:

$$w^{\tau+1} = w^{\tau} - \eta \nabla E(w^{\tau}) + \alpha w^{\tau} \tag{10}$$

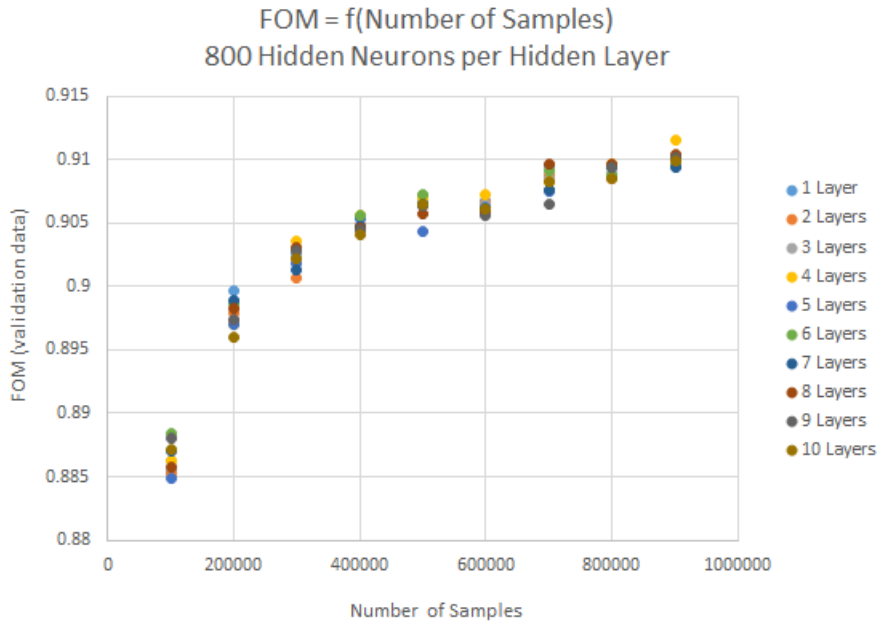

Figure 9: *The figure of merit as a function of the number of samples used, for different number of hidden layers when the number of hidden neurons per hidden layer is equal to 800.*

At the plot 10 you can see how the target on the validation data looks like and at the plot 11 you can see the distribution of the output values of the neural network. Since the results are float numbers between 0 and 1, we define that an event has passed the razor cut when the output of the NN is more than the maximum cut (see plot 12), which is most of the times 0.5.

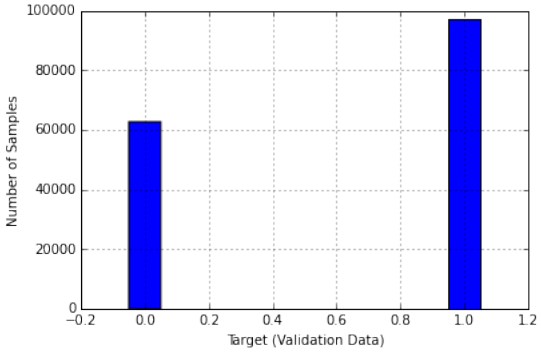

Figure 10: *Trigger Bits (TB)*

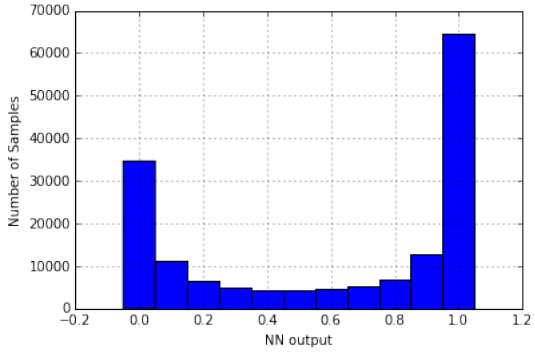

Figure 11: *Neural network output*

By trying all the different topologies and tuning the training parameters we managed to get a $\sim$ 91% figure of merit on the validation set. The percentages in each category are shown in table 1. The categories TB=0 & NN=0 and TB=1 & NN=1 are the events in match and the categories TB=1 & NN=0 and TB=0 & NN=1 are the events in mismatch.

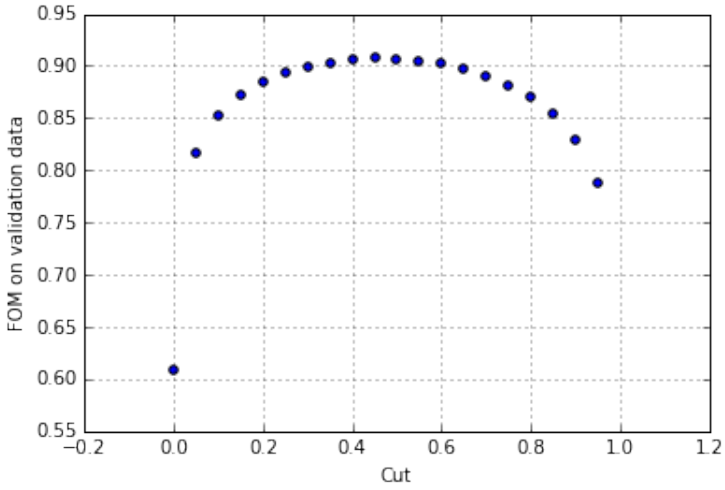

Figure 12: *FOM as a function of different cuts on the output of the NN*

Table 1: *Results*

		NN	
		0	**1**
TB	**0**	34.23%	4.93%
	1	4.35%	56.49%

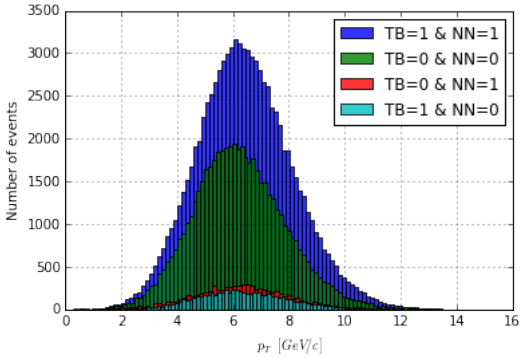

Figure 13: *p_T distribution of the jets with the highest momentum for events in match and in mismatch*

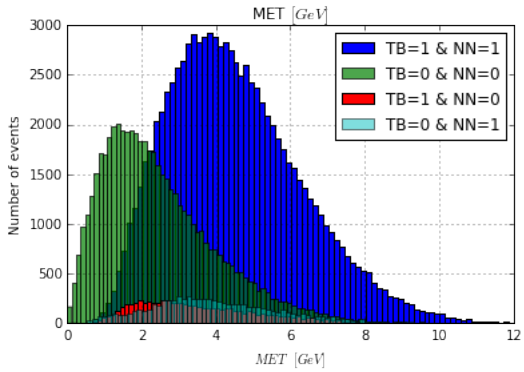

Figure 14: *MET distribution for events in match and in mismatch*

Looking at the events in mismatch ($\sim$ 10%) we see that the distribution of transverse momentum (figure 13) is the same but the distribution in MET (figure 14) is different. The green distribution, which express the events that don't pass the razor cut, looks displaced to the let compared to

the blue distribution, which express the events that pass the razor cut. The red distribution corresponds to the events in mismatch where $TB = 1$ & $NN = 0$ and behaves like the green distribution. Similarly, the light blue distribution, qhich corresponds to the events in mismatch where $TB = 0$ & $NN = 1$ behaves like the blue distribution. Therefore, it seems that network gets confused and predicts the wrong output values. The kinematic razor variable R^2 as a function of M_R is shown in figure 15 for each category. The distributions are as expected since the razor cut is defined as $R^2 M_R > 0.6$.

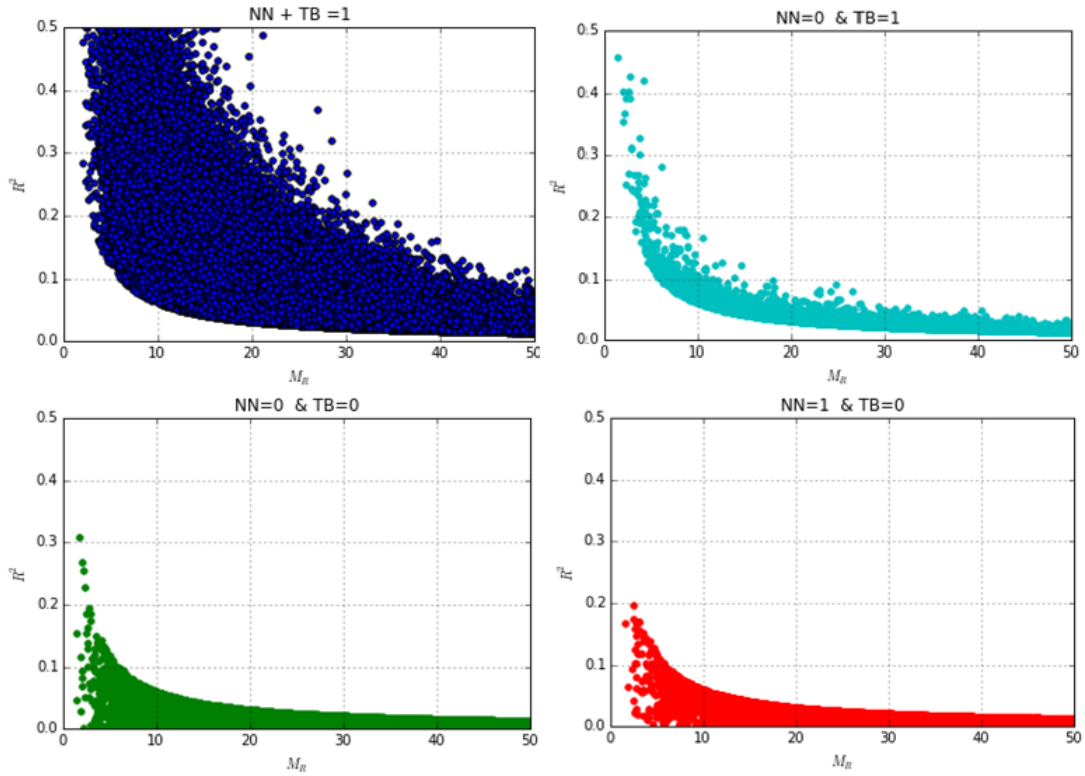

Figure 15: $R^2 = f(M_R)$

After getting 90% FOM on the razor neural network, we wanted to know if we can implement this on hardware. So, my supervisor and I contacted Sim Bamford from iniLabs in Zurich and Daniel Neil from the Institute of Neuroinformatics and the answer was positive. We can actually apply the neural network even on the L1 trigger using Neuromorphic hardware. Neuromorphic chips are electronic systems that function in a way similar to an actual brain, they are faster than GPUs or CPUs, but they can only be used with spiking Neural Networks [5] because of their time notion. Moreover, they can save power due to the fact that they focus their computational effort on currently active parts of the network. Therefore, we had to convert our neural network to a spiking neural network.

IV. Converting into a Spiking Neural Network

Deep neural networks have been the state of the art for machine learning techiques. The method used in deep neural networks is simple; there is an input vector which is given at one

time and is processed layer-by-layer. The network returns one output value. The disadvantage of deep neural networks is that they have a large computational cost, in contrast to spiking neural networks. Spiking Neural Networks (SNN) are more biologically realistic (brain-like) neural networks. They have a notion of time. The method used is that the inputs are presented as streams of events. Each spiking neuron of the SNN is composed of three computational stages: first, there is a sum for all of the neuron's input current, second, the sum is integrated over time and third, when the neuron's membrane potential raises above a potential threshold, a spike is emitted and the membrane potential resets its value to a certain reset potential.

Converting a deep neural network to a spiking neural network can cause performance loss. Some of the most important challenges in the conversion of a deep neural network to a SNN is the representation of negative values and biases. The steps for the conversion, as proposed in [6] are:

- Use rectified linear units (ReLUs) for all units of the network. A ReLU unit refers to a unit that use the activation function $max(0, x)$.
- Don't use bias during the training
- Train with back-propagation
- After training, map the weights from the ReLU network to a network of integrate-and-fire units.
- Use weight normalization for near-lossless accuracy and faster convergence.

Equally important is the right balance of spiking thresholds, input weights and input firing rates. The loss of performance can be caused by low rate, which means that the spiking unit did not receive sufficient input to cross its threshold, or by high rate, which means that the unit received too many input spikes or some of the input weights were higher than the neuron threshold.

.1 Integrate-and-fire model

The integrate-and-fire model is one of the simplest models for analyzing the behavior of neural systems. According to this model, the state of the neuron is characterized by its membrane potential. The evolution of the membrane potential, v_{mem}, is given by:

$$\frac{dv_{mem}(t)}{dt} = \sum_i \sum_{s \in S_i} w_i \delta(t - s) \tag{11}$$

where w_i is the weight of the i^{th} incoming synapse, $\delta(\cdot)$ is the delta function and $S_i = \{t_i^0, t_i^1, ...\}$ contains the spike times of the i^{th} presynaptic neuron. If the membrane potential raises above the threshold v_{res}, a spike is generated and the membrane potential is reset to a reset value u_{res}.

.2 XOR as a spiking neural network

For a start, we wanted to convert a simple neural network to a SNN. Thus, we chose the XOR neural network. The input and target values of the XOR problem are shown in table 2. First, we trained the network using Theanets, with ReLU units, no biases and with back-propagation. To simulate the spiking neural network we used the Brian [7] package. Brian is a clock-driven simulator for spiking neural networks, is easy to learn and use and is written in Python.

The parameters used for the spiking neural network are:

- Voltage threshold, v_{thr} = 950 mV
- Reset voltage, v_r =0 mV
- Simulation duration = 5 $seconds$
- Timestep dt (tau_m) =1 $second$
- Differential equation: $\frac{dv}{dt} = -v/tau_m$

Table 2: *XOR input samples and targets*

A	B	Target
0	0	0
1	0	1
0	1	1
1	1	0

The topology of the spiking neural network is as follows; the input layer is defined as a group of neurons which generates spikes with Poisson statistics and has two input values. The hidden and output layers are defined as standard groups of neurons with two and one neurons respectively. The rate is given by the $input\ values \times 6\ Hz$. The layers are fully connected and the weights in each neuron are defined as the corresponding weights from the ReLU network multiplied by 1000 mV.

In order to see the behavior of the spiking neural network, we ran the simulation for four hundred times. The results are shown in figure 16. For the first sample $(0,0)$, all the layers have zero spikes since the rate is zero. The spikes of the output layer for the other samples $(1,0)$, $(0,1)$ and $(1,1)$ have a gaussian distribution.

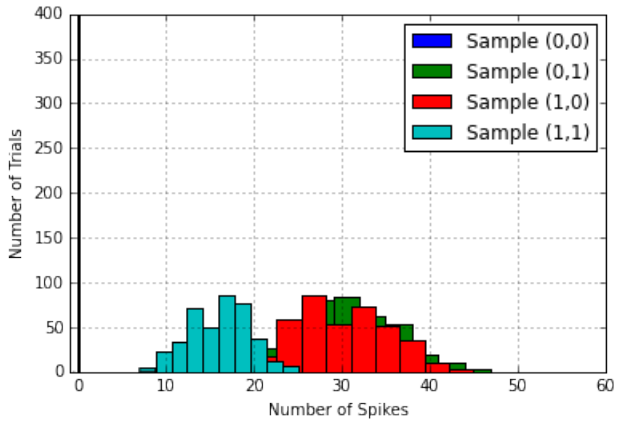

Figure 16: *The output spikes of the XOR spiking neural network, for each sample.*

As a SNN output criterion we defined that if the number of spikes in the output layer is more than 21, then the output of the SNN is equal to 1. If it's less or equal to 21, the output is 0. With this "spike" cut on the output layer, we got more than 94% agreement between the target and the SNN output in all samples. The results for each sample seperately are shown in table 3.

.3 Razor neural network as a spiking network

After the XOR example, we tried to convert the razor neural network to a spiking one. However we had some difficulties due to the weight normalization and the balancing between the spiking parameters. Unfortunately due to time constraints we didn't manage to fully convert the network to a spiking neural network. The first step was to represent the negative values of the input vector.

Table 3: *Agreement between the target and the SNN output for 400 simulations.*

Sample		Target	Agreement
0	0	0	100%
1	0	1	96%
0	1	1	94.5%
1	1	0	95%

What we did was to find the minimum negative value, and then subtract it from all the non-zero values in the input vector. Moreover, we used *L2 regularization* for the weights and changed the activation function from sigmoid to ReLU in all the hidden and output layers. Due to the new topology of the network, the FOM decreased to 85%, which is still an accepted FOM.

V. Conclusion

My project was focused on the development of a neural network which can predict if an event passes or not a razor trigger. Using synthetic data containing jets and missing transverse energy we built and trained a razor network by supervised learning. We accomplished a $\sim 91\%$ agreement between the output of the neural network and the target while the other 10% was due to the noise of the neural network. We could apply such networks during the L1 trigger using neuromorhic hardware, which are only used with spiking neural networks. Further work would be necessary to fully convert the razor network into a spiking neural network.

References

[1] Christopher Rogan. "Kinematical variables towards new dynamics at the LHC". In: *arXiv preprint arXiv:1006.2727* (2010).

[2] Leif Johnson. *Theanets documentation*. URL: `http://theanets.readthedocs.org/en/stable/`.

[3] Guido Van Rossum et al. "Python Programming Language." In: *USENIX Annual Technical Conference*. Vol. 41. 2007.

[4] Frédéric Bastien et al. *Theano: new features and speed improvements*. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop. 2012.

[5] Wolfgang Maass. "Networks of spiking neurons: the third generation of neural network models". In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[6] Peter U Diehl et al. "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing". In: ().

[7] Dan FM Goodman and Romain Brette. "The brian simulator". In: *Frontiers in neuroscience* 3.2 (2009), p. 192.