|  |  |
| --- | --- |
| Christiaan Gerhardus Viljoen | Dr Thomas Dietel |
| STUDENT NUMBER: VLJCHR004 | SUPERVISOR |
| [christiaan.viljoen@cern.ch](mailto:christiaan.viljoen@cern.ch) | [thomas.dietel@cern.ch](mailto:thomas.dietel@cern.ch) |
| Department of Statistical Sciences | Department of Physics |
| PD Hahn Building | RW James Building |
| University Avenue North | University Avenue |
| University of Cape Town | University of Cape Town |
| Rondebosch, Cape Town, South Africa | Rondebosch, Cape Town, South Africa |

# Masters Dissertation

## PHY5008W

The Application of Machine Learning Techniques towards the Optimization of High Energy Physics Event Simulations in the at

† A Large Ion Collider Experiment

‡ European Organization for Nuclear Research/ Organisation Européenne  
pour la Recherche Nucléaire

͛ Transition Radiation Detector

# Submitted in Fulfilment of the Degree: MSc Data Science



Contents

[Christiaan Gerhardus Viljoen 1](#_Toc526600342)

[Dr Thomas Dietel 1](#_Toc526600343)

[Masters Dissertation 1](#_Toc526600344)

[PHY5008W 1](#_Toc526600345)

[Submitted in Fulfilment of the Degree: MSc Data Science 1](#_Toc526600346)

[Github Repository 5](#_Toc526600347)

[Dedication 5](#_Toc526600348)

[Acknowledgements 5](#_Toc526600349)

[Plagiarism Declaration 6](#_Toc526600350)

[**MSc Candidate Full Name** 6](#_Toc526600351)

[**Student Number** 6](#_Toc526600352)

[**Faculty** 6](#_Toc526600353)

[**Department** 6](#_Toc526600354)

[**Supervisor** 6](#_Toc526600355)

[**Co-Supervisor** 6](#_Toc526600356)

[**Thesis Title** 6](#_Toc526600357)

[**DECLARATION BY STUDENT** 6](#_Toc526600358)

[**DECLARATION BY SUPERVISOR** 6](#_Toc526600359)

[Abstract 7](#_Toc526600360)

[Background 8](#_Toc526600361)

[A Large Ion Collider Experiment (ALICE) 8](#_Toc526600362)

[Introduction to the ALICE Experiment 8](#_Toc526600363)

[The ALICE Transition Radiation Detector (TRD) 8](#_Toc526600364)

[Existing Particle Physics Software 11](#_Toc526600365)

[Data Analysis Framework: ROOT 11](#_Toc526600366)

[ALIROOT 11](#_Toc526600367)

[Generative Adversarial Networks 12](#_Toc526600368)

[Introduction to Generative Adversarial Networks 12](#_Toc526600369)

[Mathematical Theory: From the Perceptron to GANs 13](#_Toc526600370)

[Application of GANS to High Energy Physics Problems 20](#_Toc526600371)

[Motivation 21](#_Toc526600372)

[Research Question 23](#_Toc526600373)

[Primary Research Question 23](#_Toc526600374)

[Secondary Research Question 23](#_Toc526600375)

[Auxiliary Research Questions 23](#_Toc526600376)

[Hypothesis 23](#_Toc526600377)

[Aims & Objectives 24](#_Toc526600378)

[Methods 26](#_Toc526600379)

[Deep Learning for Particle Identification (PID) 26](#_Toc526600380)

[Deep Learning towards Geant4 Parameter Optimization based on Environmental Conditions 26](#_Toc526600381)

[HEP Event Simulations 26](#_Toc526600382)

[Data Extraction from WLCG via AliEn 26](#_Toc526600383)

[Monte Carlo Simulation Data Generation using Geant4 with Parameters Relating to Environmental Conditions Optimized 26](#_Toc526600384)

[Generative Adversarial Networks in Keras 26](#_Toc526600385)

[Variational Autoencoders 26](#_Toc526600386)

[Productionalization of Machine Learning-Based HEP Event Simulations 26](#_Toc526600387)

[Results 27](#_Toc526600388)

[Discussion 28](#_Toc526600389)

[Conclusion and Future Work 29](#_Toc526600390)

[Bibliography 30](#_Toc526600391)

[Appendix I: ROOT/ AliEn Code 32](#_Toc526600392)

[Appendix II: Initial ML for PID 33](#_Toc526600393)

# Github Repository

A Github repository containing all files relating to this Masters Dissertation exists at:

<https://github.com/PsycheShaman/MSc-thesis>

# Dedication

This dissertation is dedicated to my mother, Elizabeth Suzanna Bloem Viljoen, who has always inspired me to follow my higher passions, despite the myriad difficulties that life makes us face; and to search fearlessly and incessantly for the deeper truths underlying our everyday world.

# Acknowledgements

Firstly, I would like to thank my father, Christiaan Gerhardus Viljoen, for all the support – material, emotional and financial – he has selflessly provided to me throughout my life, and particularly towards my higher education journey. You have no idea how much appreciation I have for all the sacrifices you have made for me, and all the advice you have given me.

Secondly, I want to thank my aunt, Professor Emma Ruttkamp-Bloem, for all the mentoring she has provided to me in navigating the world of academia, and for the inspiration that her own academic career instils in me.

Thirdly, I want to thank Dr Thomas Dietel for providing me with this immense opportunity to be part of the largest scientific experiment in human history, and for the rigorous scientific guidance that he has, and continues to provide to me.

Lastly, I would like to thank my larger family, on both my father’s and mother’s side, for providing the loving and stable environment that makes any place we assemble Home.

# Plagiarism Declaration

**TURNITIN PLAGIARISM DECLARATION**

.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MSc Candidate Full Name** | | | Christiaan Gerhardus Viljoen | | |
| **Student Number** | | | VLJCHR004 | | |
| **Faculty** | | | Science | | |
| **Department** | | | Statistical Sciences | | |
| **Supervisor** | | | Dr Thomas Dietel | | |
| **Co-Supervisor** | | | N/A | | |
| **Thesis Title** | | | The Application of Machine Learning Techniques towards the Optimization of High Energy Physics Event Simulations in the ALICE TRD at CERN | | |
| **Note: Please complete and sign the declaration.** | | | | | |
| **DECLARATION BY STUDENT** | | | | | |
| **I,** | **Christiaan Gerhardus Viljoen** | | | | (**Student’s name**) |
| hereby declare that this thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software).  Plagiarism is to use another’s work and pretend that it is one’s own and I know that plagiarism is wrong. I confirm that I have discussed and resolved any concerns emanating from the Turnitin report with my supervisor. | | | | | |
| **Signed** | |  | | **Date** |  |
|  | | | | | |
| **DECLARATION BY SUPERVISOR** | | | | | |
| **I,** | **Dr Thomas Dietel** | | | | (**Supervisor’s name**) |
| hereby declare that this thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software). Plagiarism is to use another’s work and pretend that it is one’s own. I confirm that I have discussed and resolved any concerns emanating from the Turnitin report with my student. | | | | | |
| **Signed** | |  | | **Date** |  |

# Abstract

# Background

## A Large Ion Collider Experiment (ALICE)

### Introduction to the ALICE Experiment

A Large Ion Collider Experiment (ALICE) is a large scale collaborative experiment dedicated to studying all collisions involving heavy ions at the Large Hadron Collider (LHC) at CERN (European Organization for Nuclear Research) (1).

In central high energy collisions between heavy ions (i.e. where the centres of colliding nuclei overlap sufficiently), a newly discovered deconfined state of strongly interacting matter, the Quark Gluon Plasma (QGP) can be created in small amounts (1). It is thought that this state of matter was dominant during the first 10-6 s of the Universe’s existence (2). Studying the QGP allows us to explore fundamental research avenues such as Cosmology, the Evolution of our Universe, and one of the fundamental forces in the standard model that is the hardest to probe: the strong nuclear force (2).

ALICE is the first experiment in history capable of producing the QGP in a laboratory setting; and as such, it is equipped to infer a variety of physical variables relating to the QGP, by analysing data from electrons produced during many of the physical processes that occur in the wake of heavy ion collisions, e.g. open heavy-flavour hadron decays, virtual photons, etc (1). Robust electron identification is therefore a crucial part of studying the QGP, and accurately-tuned detector triggers are required to ensure the collection of sufficient amounts of data to guide inferences regarding the statistical distributions of the abovementioned measurables (1).

## The ALICE Detector Set-Up

The ALICE apparatus’ dimensions are and weighs around 10 000 tonnes and consists of 17 detector systems, categorized into three sections: central barrel detectors that are immersed in a 0.5 Tesla magnetic field and are used to detect particles produced at midrapidity (the TRD, whose output will be simulated in this project, is located within the central barrel), forward detectors which are important in triggering, investigating multiplicity and event characterization and the MUON spectrometer, which measures muons of high transverse momentum , quarkonium and light vector mesons (3).

### The ALICE Transition Radiation Detector (TRD)

#### Introduction to the TRD

The main purpose of the ALICE Transition Radiation Detector (TRD) is the identification of electrons, as well as the operation of event triggers that determine whether data from a specific collision should be kept, based on measurements such as collision centrality, amongst others. As an added benefit, the TRD informs the ALICE central barrel’s calibration, and the data it produces is used extensively during track reconstruction and particle identification (1).

#### TRD Physical Properties

The TRD is located within the ALICE central barrel, it’s inner boundary sits at a radial distance of 2.90 m from the beam axis. It contains 522 chambers, each of which contains a fiber/ foam radiator, a 3 cm drift region and a multi-wire proportional chamber (MWPC) filled with Xenon-; these chambers are arranged in a six-layer configuration (1). The position of the TRD within the ALICE central barrel is shown in *Figure 1*.

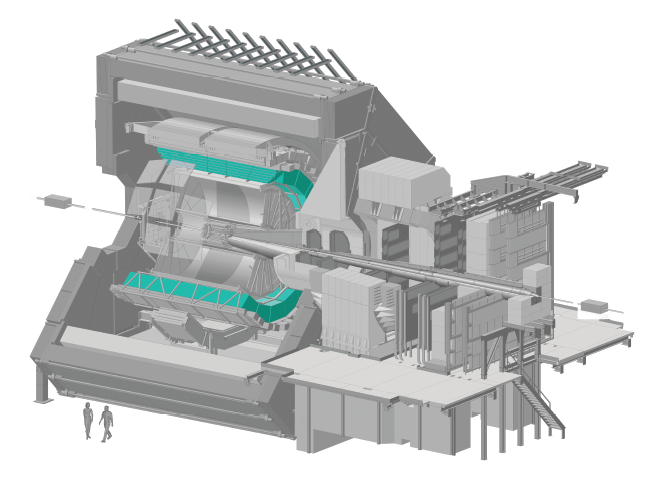


Figure 1: The ALICE TRD, Highlighted in Cyan, within the ALICE main detector (4)

#### TRD Measurement Mechanism

As the name suggests, transition radiation occurs when a particle transits across a dielectric boundary, this radiation is often measured in particle detectors to inform track reconstruction. Multiple boundaries are typically required to increase radiation yield, and since highly relativistic particles emit transition radiation that extends into the X-ray domain, the TRD utilizes gases with high proton-number (Z) to absorb this radiation, resulting in a high yield of energy deposition relative to the energy lost via ionization (1).

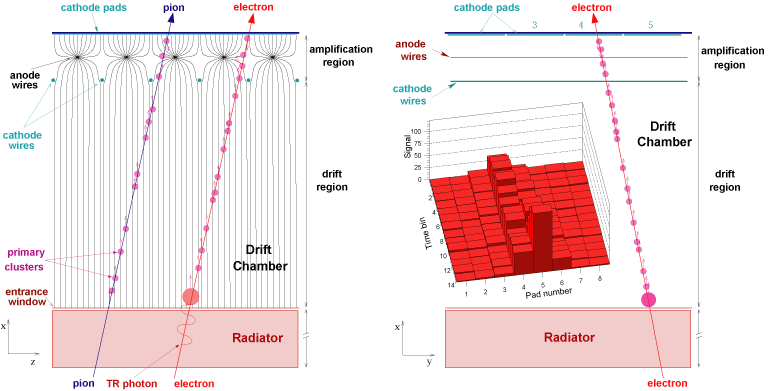


Figure 2: A schematic representation of the components in an MWPC Module (5)

The drift time of gas particles within the MWPC provides fine-grained positional information about where the particle tracklet passed through the radiator. The detected signal takes the form of charged gas molecules (ionized via interaction with transition radiation photons and amplified through a chain of interactions between gas molecules), finally being absorbed by a negatively charged wire (anode) (4), this process is depicted in *Figure 2*.

#### Identifying Electrons

The production of Transition Radiation (TR), as well as a higher specific ionization energy loss , are two features that enable accurate differentiation of electrons from other charged particles; and the temporal data provided by the TRD further enhances the specificity and sensitivity of the electron identification process (1).

On average, a single TR photon with an energy range of 1-30 is produced by an electron with momentum higher than 1 .

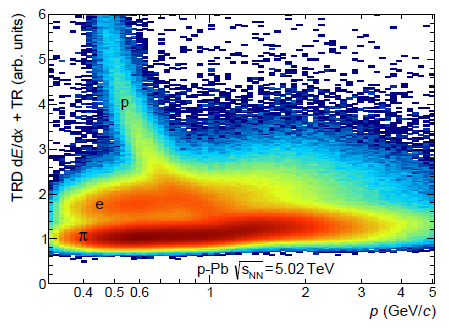


Figure 3: sum of TRD signal (Transition radiation plus energy loss due to ionization), as a function of momentum, for Protons (), Electrons () and Pions () (3)

During particle identification (PID), the signal from each chamber is sliced into 7 conceptual sections, moving inward from the outermost readout, each section integrates the signal captured in ±5 mm of detector thickness, see ***Figure 4*** for an indication of how the electron signal (scaled by the pion signal) decays in sections further out from the detector entrance, since most TR is absorbed in the sections closer to the detector entrance (3).

The main analytical methods used to identify electrons are TPC and TOF particle identification and topological cuts (3).

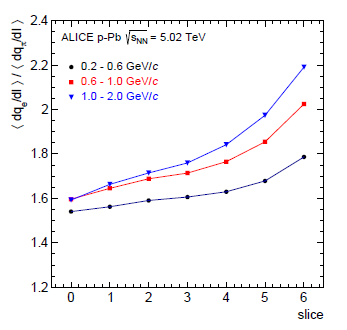


Figure 4: The average signal ratio (signal units ), as a function of slice number. larger slice numbers achieve a higher electron signal, since they are closest to the entrance of the detector, where most TR is absorbed (3)

#### Current TRD Accuracy

The TRD detector performance is quantified by pion efficiency, which is the proportion of pions misclassified as electrons (i.e. the False Positive Rate). The pion rejection factor is the inverse of this, i.e. (3).

Researchers are also interested in the electron efficiency (i.e. the True Positive Rate), since the process of eliminating pions will inevitably result in some elimination of electrons misclassified as pions; so ideally one would want to maximize electron efficiency, whilst simultaneously maximizing the pion rejection factor.

Currently, at a momentum of around 1 GeV/*c*, a pion rejection factor of 410 is achievable in p-Pb (proton-Lead) collisions, with resolution improving by about 40% when TRD data is included in track reconstruction (1), with electron efficiency achieved in this momentum range being around 90% [citation needed, inferred from a plot in this paper (3)].

## Existing Particle Physics Software

### Data Analysis Framework: ROOT

ROOT is an open-source data analysis framework developed by the High Energy Physics (HEP) community; it is an object oriented platform, written in C++ and allows for storage of classes in a compressed binary file format (.root), as well as an object container class (TTree), which facilitates statistical analysis of large datasets via vertical storage methods, and enabling analysis across files on local disks, shared file systems, and the Worldwide LHC Computing Grid (WLCG) (6).

### ALIROOT

AliROOT is an extension of ROOT, used specifically by the ALICE collaboration. It is an integrated ecosystem consisting of event generators (PYTHIA6, HIJING, DPMJET and ISAJET), Virtual Monte Carlo simulators (Geant3, Geant4 and Fluka) and various steering classes for simulation and reconstruction; all of which sits upon the basic ROOT architecture, and which communicates via the ALICE Environment Grid Framework (AliEn) with the WLCG. This architecture is depicted in *Figure3* (7)*.*

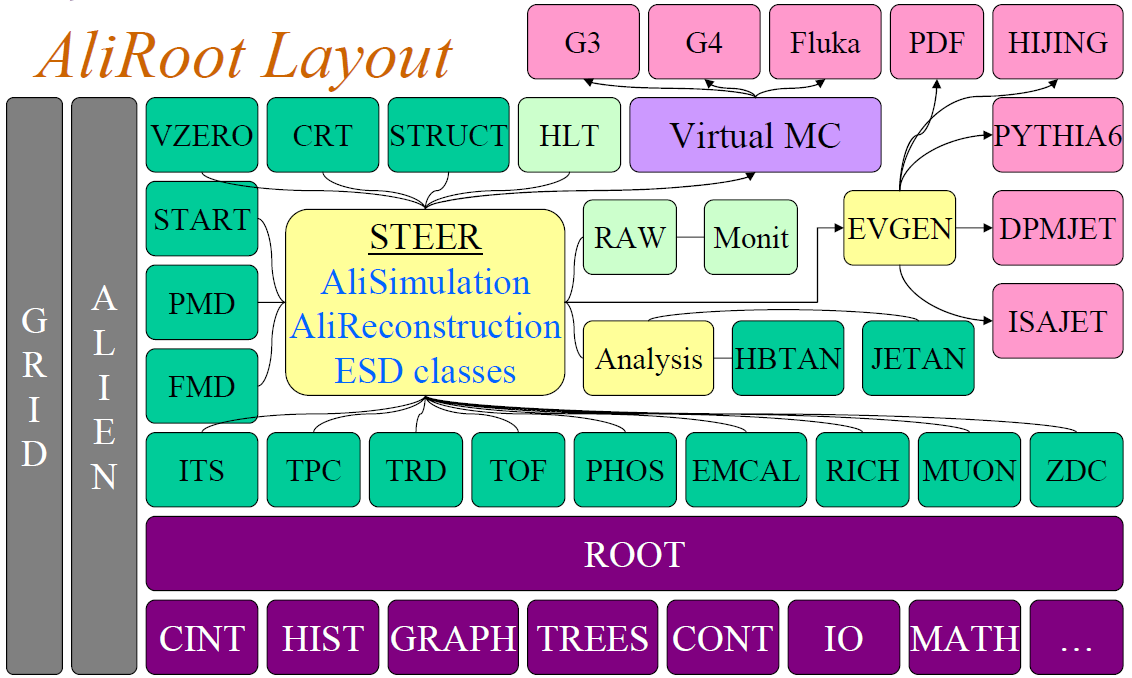


Figure 5: ALIROOT Ecosystem (7).

#### Integration with other Programming Languages

PyROOT and ROOT R are interfaces that allow seamless integration between ROOT and Python as well as R, respectively. Features include calling functions from any Python or R package within ROOT, translating datatypes between languages, as well as access to the R graphical system from within ROOT (8).

## Generative Adversarial Networks

### Introduction to Generative Adversarial Networks

In 2014, Goodfellow et. al. (9) proposed a novel implementation of Deep Learning, involving an adversarial mini-max game between two Artificial Neural Networks (ANNs), which they called Generative Adverserial Nets (GANs). A GAN model consists of two elements: a generative model G, and a discriminative model D. During training, G aims to maximize the loss function of D by accurately capturing the underlying distribution of the training data, and D estimates the probability of whether an observation fed to it is “real”, i.e. it is from the actual data distribution or “fake”, i.e. it was generated by G (9).

If both models are ANNs, they can be trained simultaneously via backpropagation, until a unique solution can theoretically be reached, where G accurately models the data distribution and D outputs 0.5 everywhere (9).

### Mathematical Theory: From the Perceptron to GANs

#### The Perceptron

The field of Deep Learning is based on the idea of a “perceptron” proposed by Rosenblatt in 1958 (10), originally envisioned as a way to mathematically model neural information processing in the brain, it was not very successful in capturing the complexity inherent in neurobiological information processing, but very useful for learning non-linear patterns in data. ***Figure 4*** informs the mathematical theory that follows.

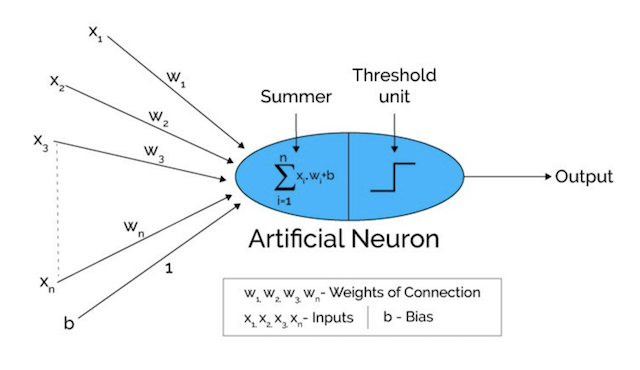


Figure 6: A Schematic Representation of a Single Perceptron (11)

In the figure above an observation , consisting of a vector of input features , … , is mapped to an estimated outcome variable: , by:

1. Multiplying each element of the input vector with the transpose of a corresponding vector of weights, i.e. of the same length as X, i.e. , …,
2. Summing the individual products of all the input features with their corresponding weights, and adding a bias term[[1]](#footnote-1):

Passing the result of step 2 (i.e. Z) through an “Activation Function” , where the function , is typically one of:

* 1. Rectified Linear Unit (ReLU):
  2. Sigmoid function:
  3. Hyperbolic Tangent (Tanh):

The result of is an estimate () for the outcome variable . And a simple cost function, such as the root mean squared error:

RMSE = ,

can be minimized via gradient descent to find the optimal values for and .

The original motivation for the use of the activation function is inspired by the way a biological neuron is triggered: it receives inputs from various other neurons, sums up their inputs and fires once a threshold is reached. Practically, in a deep learning set up non-linearities in the data can be captured by these activation functions, particularly when perceptrons are chained into multiple fully-connected layers, as is the case in Deep Learning, which we’ll discuss next.

#### Deep Learning

Deep Learning extends the concept of a single perceptron, by streaming a matrix of training observations (each of which is a vector of features, e.g. ) through multiple layers of perceptrons stacked together into what is called a hidden layer architecture.

The output of each layer’s activation function becomes the input vector fed to the next layer, i.e. for = hidden layers:

Where is the result of the first activation function, i.e.

##### A Note on Notation:

The initial matrix of inputs can be denoted as , and the output of each subsequent layer’s activation function in the nested equation can also be written in notation.

#### Visual Intuition

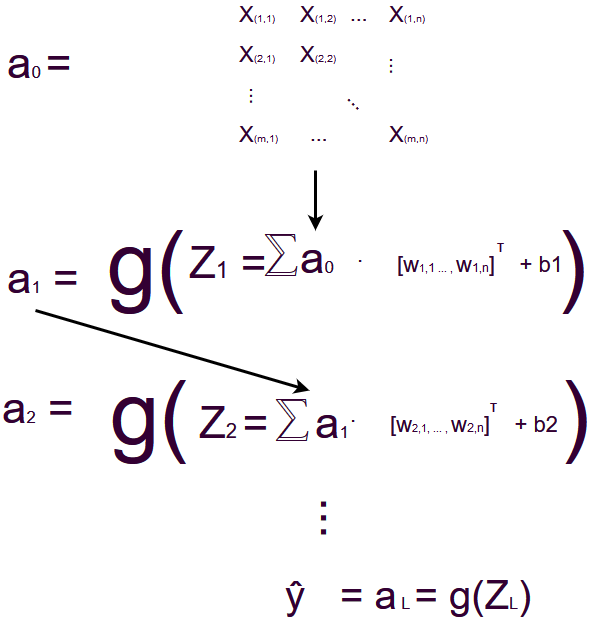


Figure 7: A visual Representation of the mathematics Explained Above (12)

#### Backpropagation

Backpropagation is an iterative process of adjusting weights and biases, by minimizing a cost function , which can be understood as the error in the estimate tied to a specific value of weights and biases.

Each element of the weights and biases are typically updated proportionately to the partial derivative of the cost function with respect to that specific element, multiplied by a learning rate , i.e.

For the element of the weight matrix , is updated as:

Similarly, for the element of the bias vector b, is updated as:

The training dataset is fed through the network multiple times (called epochs), hopefully allowing the cost function to reach a minimum.

#### Hyperparameters

To arrive at an optimal neural network for the defined task, there are multiple hyperparameters that can be optimized. Some of these have already been mentioned in part, i.e. the number of epochs, the choice of activation function, the number of hidden layers and the number of perceptrons within each hidden layer, the distribution of weights to initialize the network with, the learning rate ,etc.; but there are many more, including regularization, dropout, etc., which will be covered in the implementation section of this thesis.

#### Generative Neural Networks Explained at the hand of Pseudocode, based on a minimal example from (13)

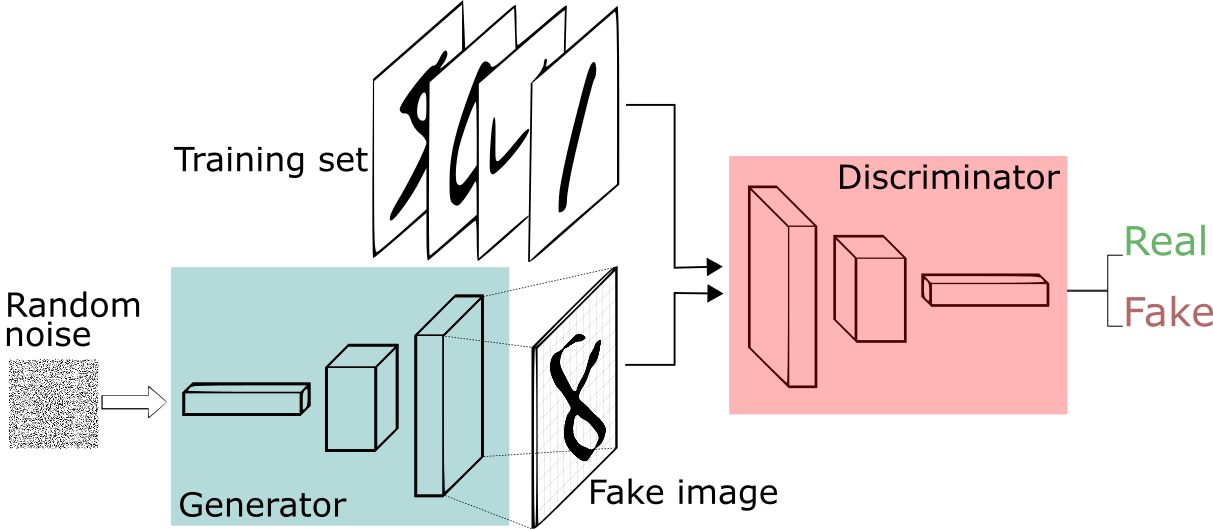


Figure 8: Schematic Representation of a Generative Adverserial Network (14)

Suppose we have a set of training images, each of which consists of 256 pixels, arranged in 28 rows and 28 columns; such as the well-known MNIST database of handwritten images (15), of which a sample is shown in ***Figure 7*** below:

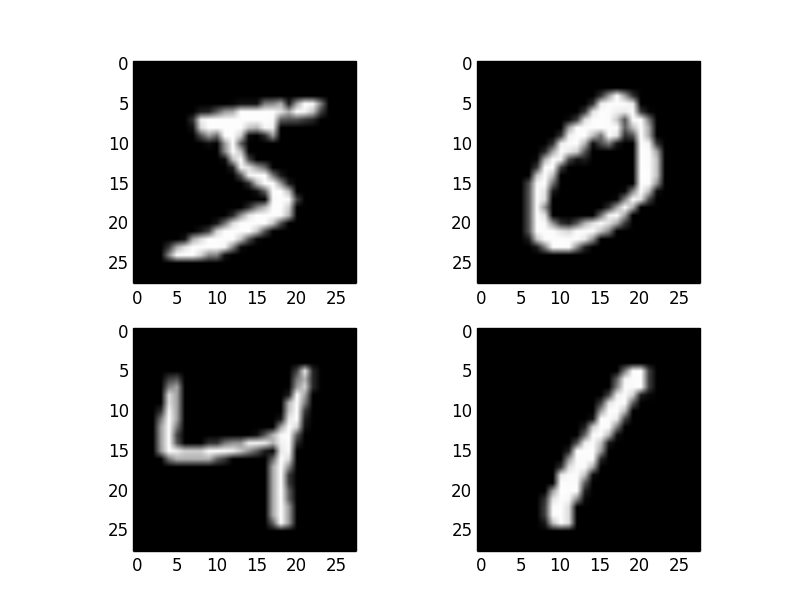


Figure 9: a sample of four handwritten digits from The MNIST Dataset (16)

Our goal is to build a GAN architecture, using these images. GANs is explained in the following lines of pseudocode, with comments indicated in bold text, preceded by a double forward-slash (“//”):

##### Pseudocode

**//DEFINE THE SHAPE OF THE INPUT (AND THEREFORE OUTPUT) IMAGES:**

IMAGE.SHAPE = [ROWS=28,COLUMNS=28]

**//GENERATE A VECTOR RANDOM GAUSSIAN NOISE TO USE AS INPUT TO THE GENERATOR NETWORK:**

A\_0 = RANDOM.NOISE(SIZE=100)

**//DEFINE THE HIDDEN ARCHITECTURE OF THE GENERATOR FUNCTION:**

GENERATOR = SEQUENTIAL\_MODEL[

**//Z\_1 IS THE INPUT LAYER, WITH THE SAME DIMENSIONS AS THE RANDOM NOISE INPUT VECTOR**

Z\_1 = LAYER[NODES=256, INPUT = A\_0]

**//A\_1 IS THE RESULT OF APPLYING A “LEAKY RELU” ACTIVATION FUNCTION TO:**

**//Z\_1 = SUM(W\_T \* A\_0 + BIAS)**

A\_1 = LEAKY\_RELU(ALPHA = 0.2,

**//LEAKY RELU ACTIVATION IS AN INDICATOR FUNCTION,**

**DEFINED AS FOLLOWS:**

FUNCTION = IF Z\_1 > 0 THEN Z\_1 ELSE A\*Z\_1 END,

INPUT = Z\_1)

**//BATCH NORMALIZATION IS A WAY OF SCALING THE TRAINING SAMPLE FEAUTURES, BY REDUCING COVARIANCE SHIFT, AND AS A RESULT, SPEEDING UP LEARNING AND PREVENTING VANISHING OR EXPLODING GRADIENTS DURING BACKPROPAGATION TRAINING**

A\_1 = BATCH\_NORMALIZATION(A\_1)

**//A FURTHER TWO HIDDEN LAYERS ARE ADDED, EACH TIME DOUBLING THE AMOUNT OF PERCEPTRONS PER LAYER, AND APPLYING AN ACTIVATION FUNCTION, AS WELL AS BATCH NORMALIZATION TO EACH LAYER**

Z\_2 = LAYER[NODES=512, INPUT = A\_1]

A\_2 = LEAKY\_RELU(ALPHA = 0.2,

FUNCTION = IF Z\_2 > 0 THEN Z\_2 ELSE ALPHA\*Z\_2 END,

INPUT = Z\_2)

A\_2 = BATCH\_NORMALIZATION(A\_2)

Z\_3 = LAYER[NODES=1024, INPUT = A\_2]

A\_3 = LEAKY\_RELU(ALPHA = 0.2,

FUNCTION = IF Z\_3 > 0 THEN Z\_3 ELSE ALPHA\*Z\_3 END

INPUT = Z\_3)

A\_3 = BATCH\_NORMALIZATION(A\_3)

**//FINALLY, THE SECOND HIDDEN LAYER IS RESHAPED TO THE ORIGINAL DIMENSIONS (28\*28), AND A TANH ACTIVATION FUNCTION (AS DEFINED EARLIER IN THE MATHEMATICAL THEORY SECTION OF THIS THESIS) IS APPLIED TO IT**

Y\_HAT = TANH(A\_3.RESHAPE(IMAGE.SHAPE))

]

DISCRIMINATOR = ANOTHER NEURAL NETWORK OR OTHER FORM OF DISCRIMINATOR

For a defined number of epochs, a sample of both real (training example) and fake (generated by G) images are streamed through the Discriminator Network (D). The loss function of the D informs backpropagation through both networks.

## Autoencoders

Autoencoders are an implementation of artificial neural networks (ANNs) that map from a full representation of an input to a compressed representation (encoding), and then maps back from the compressed representation to the original input (decoding). It contains a hidden layer which learns a latent representation of the input, and the network as a whole consists of an encoder function and a decoder function that results in a reconstruction (17), see ***Figure 10***, below.

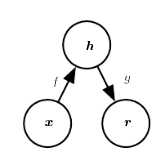


Figure 10: The General structure of an autoencoder, illustrating how the hidden representation layer (h) is inferred from the input data by f(x); and how the input data is subsequently reconstructed by g(h) (17)

Autoencoders are restricted so that they do not copy the input data exactly, to enable them to learn useful representations of the input data, i.e. the mapping functions are not deterministic, but contain a degree of stochasticity in their mapping protocol (17).

Since AEs are an extension of feedforward neural networks, they can similarly be trained using back-propagation based on mini-batch gradient descent, but unlike traditional feedforward ANNs, training can also be accomplished using recirculation (a method that compares activations in the encoding function to those in the reconstructing function) (17).

It is important to note that the output of the decoder function is generally not the end-goal of implementing an AE, but that arriving at a useful structure of , which encodes important features from the input data, is what motivates the usage of this algorithm. To this end, can be constrained to a smaller feature-space compared to , i.e. an undercomplete representation, which hopefully captures the most important latent variables that determine variability in .

Training an AE involves minimizing a loss function , which penalizes proportionally to the dissimilarity of its output to the original input (17).

### Variational Autoencoders (VAEs)

## Application of Traditional Deep Learning to High Energy Physics Problems

## Application of Generative Models to High Energy Physics Problems

# Motivation

## High Energy Physics Simulations

Detector response simulations is an integral part of all HEP experiments and high accuracy is attainable with first principle methods in current simulation environments such as GEANT4, but at a high computational cost (18).

This computational cost is particularly relevant to the ALICE experiment, where a much higher charged particle multiplicity is achieved, relative to average transverse momentum . In other words, many more particles are produced in heavy ion collisions () compared to proton-proton () collisions, which increases simulation time significantly (19) 🡨placeholder reference, since I am just guessing here... , see ***Figure 8*** below

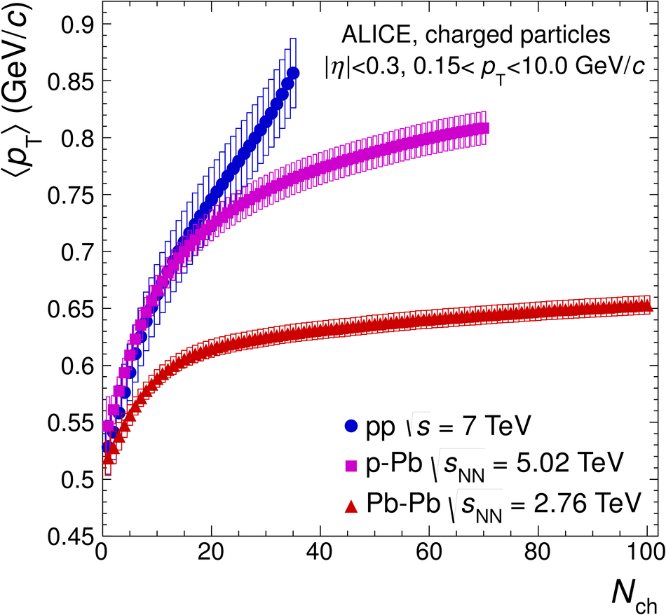


Figure 11: Average Charged Particle Multiplicity vs Average Transverse Momentum in p-p, p-Pb and Pb-Pb collisions at the LHC (20)

# Research Question

## Primary Research Question

To what level of accuracy can generative machine learning algorithms, such as Generative Adversarial Networks and Variational Autoencoders be developed and trained, in order to simulate data generated by the ALICE TRD during High Energy Physics event simulations?

## Secondary Research Question

Can trained weights be sensibly extracted from individual neurons, or from hidden layers, of a Generative Neural Network, in order to infer:

* the latent variables discovered by the algorithm,
* their relative importance in outputting a detector simulation,
* and to what extent these latent variables map back to observable physical variables and processes, currently used in Monte Carlo simulations, based upon first principles

## Auxiliary Research Questions

1. Can an ensemble of Machine Learning Algorithms be employed successfully to optimize parameters in the existing Monte Carlo event simulation environment, Geant4?
2. Could the Discriminative Neural Network be replaced by a different classification algorithm, such as a support vector machine or an ensembled method, in order to arrive at a more accurate GAN architecture?
3. Could the random gaussian noise-source (z), used by G to simulate detector data, be replaced with a more sensible set of features from the original collision conditions, such as center of mass energy, environmental conditions, etc.

# Hypothesis

It is the hypothesis of this dissertation that Generative Machine Learning Algorithms, such as GANs, can be successfully be applied to HEP event simulations, at a lower computational cost than traditional methods currently being used, and that these algorithms can output data which is indistinguishable from actual data collected from the TRD at CERN.

# Aims & Objectives

1. **To build a highly accurate Neural Network that is able to classify particles passing through the TRD as: electrons, positrons, pions, etc.**
   1. By using h2o.ai (21) within the R statistical software environment (22)
2. **To optimize parameters for Monte Carlo event simulations within Geant4, in order to more accurately account for environmental conditions in the TRD at run-time, e.g. ambient temperature, atmospheric pressure, etc.,**
   1. using an ensembled approach of machine learning (ML) algorithms within h2o.ai (21) ecosystem
3. **To simulate Particle-Detector Interaction data,**
   1. By modelling the output data generated during High Energy Physics Collisions in the ALICE TRD,
   2. that is of sufficient quality so as to be indistinguishable from data generated by current Monte Carlo simulations, such as that generated by Geant4
   3. and that is indistinguishable from data taken from real collision events within the ALICE TRD at CERN
4. **To build an efficient “Proof of Concept” Generative Adversarial Network architecture to this end (point 3.),**
   1. By utilizing existing packages for Deep Learning, e.g. Keras for proof of concept, within the R statistical software environment,
   2. Using data from:
      1. real HEP experiments at ALICE,
         1. Which will be obtained from the WLCG storage system using AliEn, and parsed (using AliRoot) into a data format (.csv/ .json) that can be read into R
      2. Simulated event data from Geant4, with parameters tuned to emulate the effect of environmental variables as mentioned in point 2., above.
5. **To explore variational autoencoders (VAEs) as an alternative methodology for event simulations**
6. **To productionalize the most accurate ML simulations of event data (GANs, VAEs, an ensemble of the two, or something completely different)**
   1. Either by:
      1. recoding the chosen algorithm in C++, based upon first principles from linear algebra outlined in the Mathematical Theory section in the Background of this document; and utilizing the existing ROOT package for ML, the Toolkit for Multivariate Data Analysis (TMVA) to support the implementation of this
      2. Interfacing with ROOT from within R, using ROOT R, and setting up a RESTful API service, using the plumbeR package, to minimize additional dependencies at run time

# Methods

## Deep Learning for Particle Identification (PID)

## Deep Learning towards Geant4 Parameter Optimization based on Environmental Conditions

## HEP Event Simulations

### Data Extraction from WLCG via AliEn

### Monte Carlo Simulation Data Generation using Geant4 with Parameters Relating to Environmental Conditions Optimized

### Generative Adversarial Networks in Keras

### Variational Autoencoders

## Productionalization of Machine Learning-Based HEP Event Simulations

# Results

# Discussion

# Conclusion and Future Work

# Bibliography

1. *The ALICE Transition Radiation Detector: Construction, operation and performance.* **Collaboration, ALICE.** 2018, Nuclear Inst. and Methods in Physics Research, Vol. 881, pp. 88-127.

2. **Reed, Rosi.** What is the QGP? *Quantum Diaries.* [Online] [Cited: 3 October 2018.] https://www.quantumdiaries.org/2011/11/23/what-is-the-qgp/.

3. *Performance of the ALICE Experiment.* **ALICE Collaboration.** s.l. : CERN-PH-EP, 2014, Vol. 031.

4. **More details on the ALICE TRD. *ALICE - CERN.* [Online] CERN. [Cited: 3 October 2018.] http://alice.web.cern.ch/detectors/more-details-alice-trd.**

**5. The Transition Radiation Detector. *ALICE.* [Online] [Cited: 4 October 2018.] http://aliceinfo.cern.ch/Public/Objects/Chapter2/DetectorComponents/trd\_operation.gif.**

**6. *ROOT -- A C++ framework for petabyte data storage, statistical analysis and visualization.* Antcheva, I., et al. 12, 2009, Computer Physics Communications, Vol. 180, pp. 2499-2512.**

**7. ALICE Collaboration. AliRoot Tutorial. *CERN Indico.* [Online] [Cited: 4 October 2018.] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=2ahUKEwi748CmwezdAhVSasAKHUVkDwsQFjABegQICBAC&url=https%3A%2F%2Findico.cern.ch%2Fevent%2F6986%2Fcontribution%2F0%2Fattachments%2F1029652%2F1466277%2Ftutorial.pdf&usg=AOvVaw.**

**8. Integration with other languages. *ROOT.* [Online] [Cited: 4 October 2018.] https://root.cern.ch/integration-other-languages.**

**9. *Generative Adverserial Nets.* Goodfellow, Ian J, et al. 2014, stat.ML.**

**10. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* Rosenblatt, F. 6, 1958, The Psychological Review, Vol. 65.**

**11. From Perceptron to Deep Neural Nets. *Becoming Human AI.* [Online] [Cited: 4 October 2018.] https://becominghuman.ai/from-perceptron-to-deep-neural-nets-504b8ff616e.**

**12. Viljoen, Christiaan Gerhardus. *Created on draw.io.***

**13. Keras GAN. *Github.* [Online] [Cited: 4 October 2018.] https://github.com/eriklindernoren/Keras-GAN/blob/master/gan/gan.py.**

**14. A Beginner's Guide to Generative Adverserial Networks (GANs). *Skymind.* [Online] [Cited: 4 October 2018.] https://skymind.ai/wiki/generative-adversarial-network-gan.**

**15. LeCunn, Yan, Cortes, Corinna and Burges, Christopher. The MNIST Database of Handwritten Images. *Yan Lecunn.* [Online] http://yann.lecun.com/exdb/mnist/.**

**16. Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras. *Machine Learning Mastery.* [Online] https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/.**

**17. Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron. *Deep Learning.* s.l. : The MIT Press, 2016. ISBN 9780262035613.**

**18. *Generative Models for Fast Cluster Simulations in the TPC for the ALICE Experiment.* Deja, Kamil, Trzcinski, Tomasz and Graczykowski, Lukasz.**

**20. *Multiplicity dependence of the average transverse momentum in pp, p-Pb, and Pb-Pb collisions at the LHC.* 2013, Phys. Lett., pp. 371-380.**

**21. H2O.ai.**

**22. R Core Team. R: A language and environment for statistical Computing. Vienna : s.n.**

**23. Karazeev, Anton. Generative Adversarial Networks (GANs): Engine and Applications. *Stats and Bots.* [Online] [Cited: 6 October 2018.] https://blog.statsbot.co/generative-adversarial-networks-gans-engine-and-applications-f96291965b47.**

# Appendix I: ROOT/ AliEn Code

# Appendix II: R code

## Initial ML for PID

**Set up session for Deep Learning:**

Set working directory:

#setwd("~/Documents/R")

Clear R environment objects:

rm(list=ls())

Read in JSON dump of Chris’s python dictionary and convert to three dataframes:

require(jsonlite)  
  
dat <- fromJSON("data/text.txt")  
  
zero <- dat[[1]]  
one <- dat[[2]]  
two <- dat[[3]]  
three <- dat[[4]]

Union these dataframes into one:

all.dat <- data.frame(  
 rbind(  
 zero,  
 one,  
 two,  
 three  
 )  
)

**Data Wrangling for Deep Learning:**

require(dplyr)  
library(tidyr)  
  
#isolate the layer columns  
layers <- data.frame(all.dat[,c(8,12,16,20,24,28)])  
  
#choose an example that is not null  
fix <- layers[6,1]  
  
#fill it with zeros, retaining its shape  
for(i in 1:nrow(fix[[1]])){  
 for(j in 1:ncol(fix[[1]])){  
 fix[[1]][i,j] <- 0  
 }  
}  
  
#replace any null values with this matrix of zeroes  
nullToNA <- function(x) {  
 x[sapply(x, is.null)] <- fix  
 return(x)  
}  
  
layers <- as.matrix(layers)  
layers <- nullToNA(layers)  
  
layers <- cbind(all.dat[,1:3],layers)  
  
#unnest each layer individually:  
#//TODO: create a function for this  
  
#LAYER 0  
  
a <- rep(NA,264)  
  
for(i in layers$layer0){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(all.dat[,c(1:7,9:11,13:15,17:19,21:23,25:27)],a)  
  
#LAYER 1  
  
a <- rep(NA,264)  
  
for(i in layers$layer1){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 2  
  
a <- rep(NA,264)  
  
for(i in layers$layer2){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 3  
  
a <- rep(NA,264)  
  
for(i in layers$layer3){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 4  
  
a <- rep(NA,264)  
  
for(i in layers$layer4){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 5  
  
a <- rep(NA,264)  
  
for(i in layers$layer5){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#make all missing values 0  
layer0[is.na(layer0)] <- 0  
  
my.dat <- layer0[,4:100]  
  
#make the outcome variable categorical  
my.dat$pdgCode <- as.factor(my.dat$pdgCode)  
  
#save the manipulated dataset and clear the R environment  
save(my.dat,file="NNdata.RData")  
rm(list=ls())

Scale the wrangled data and remove unnecessary elements:

#load the manipulated data set  
load("NNdata.RData")  
#scale numerical predictor variables  
my.dat[,-1] <- scale(my.dat[,-1])  
my.dat <- my.dat %>%  
 subset(pdgCode!=-11) %>%  
 subset(pdgCode!=11)

**DEEP LEARNING:**

require(h2o)  
h2o.init(max\_mem\_size = "28G",nthreads = -1)

h2o.removeAll()

*h2o.no\_progress()*

Upload data to H2O cluster, split into training (60%), validation (20%) and test (20%) sets:

dat.hex <- as.h2o(my.dat,"dat.hex")

splitz <- h2o.splitFrame(dat.hex,ratios=c(0.6,0.2),  
 destination\_frames = c("train.hex","valid.hex","test.hex"))  
  
train.hex <- splitz[[1]]  
valid.hex <- splitz[[2]]  
test.hex <- splitz[[3]]

Build first neural network:

* 2 hidden layers of 200 neurons each
* 100 epochs
* 10-fold cross validation

nn\_1 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_1",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 hidden = c(200,200),  
 nfolds=10,  
 standardize = F,  
 epochs=100,  
 fast\_mode = F,  
 sparse = T  
)

Build a differenct architecture:

* One hidden layer of 500 neurons
* 100 epochs
* 10-fold cross validation

nn\_2 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_2",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 hidden = 500,  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=100  
)

Build yet another differenct architecture:

* 10 hidden layers
* 100 epochs
* 10-fold cross validation

nn\_3 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_3",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 hidden = c(32,32,32,32,32,32,32,32,32,32),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=100  
)

Find most probable architecture:

h2o.mean\_per\_class\_error(nn\_1)

## [1] 0.2241379

h2o.mean\_per\_class\_error(nn\_2)

## [1] 0.2155172

h2o.mean\_per\_class\_error(nn\_3)

## [1] 0.2241379

h2o.confusionMatrix(nn\_1)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.0230669511644981:  
## -211 211 Error Rate  
## -211 32 26 0.448276 =26/58  
## 211 0 66 0.000000 =0/66  
## Totals 32 92 0.209677 =26/124

h2o.confusionMatrix(nn\_2)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.120827021588128:  
## -211 211 Error Rate  
## -211 33 25 0.431034 =25/58  
## 211 0 66 0.000000 =0/66  
## Totals 33 91 0.201613 =25/124

h2o.confusionMatrix(nn\_3)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.566688777517043:  
## -211 211 Error Rate  
## -211 32 26 0.448276 =26/58  
## 211 0 66 0.000000 =0/66  
## Totals 32 92 0.209677 =26/124

h2o.confusionMatrix(nn\_1,newdata = test.hex)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.416494562377328:  
## -211 211 Error Rate  
## -211 6 10 0.625000 =10/16  
## 211 1 20 0.047619 =1/21  
## Totals 7 30 0.297297 =11/37

h2o.confusionMatrix(nn\_2,newdata = test.hex)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.120827021588128:  
## -211 211 Error Rate  
## -211 4 12 0.750000 =12/16  
## 211 0 21 0.000000 =0/21  
## Totals 4 33 0.324324 =12/37

h2o.confusionMatrix(nn\_3,newdata = test.hex)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.566688777517043:  
## -211 211 Error Rate  
## -211 6 10 0.625000 =10/16  
## 211 0 21 0.000000 =0/21  
## Totals 6 31 0.270270 =10/37

plot(nn\_1)



plot(nn\_2)



plot(nn\_3)



**Tuning:**

The above information suggests fewer hidden layers, with more neurons is more suited to this problem.

We build a fourth neural network architecture: \* 2 hidden layers of 500 neurons each \* 200 epochs

And test this with various activation functions:

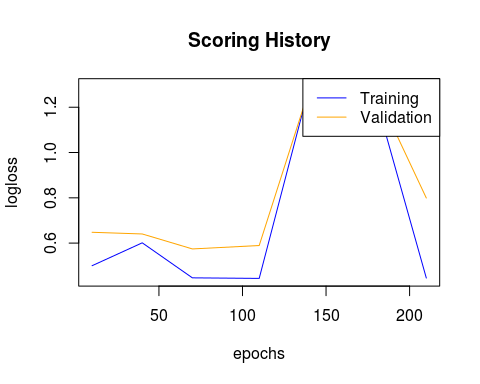
**Tanh with Dropout**

nn\_4 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_4",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "TanhWithDropout",  
 hidden = c(500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T  
)

h2o.performance(nn\_4,test.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.3263933  
## RMSE: 0.5713084  
## LogLoss: 1.019133  
## Mean Per-Class Error: 0.34375  
## AUC: 0.6845238  
## Gini: 0.3690476  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 5 11 0.687500 =11/16  
## 211 0 21 0.000000 =0/21  
## Totals 5 32 0.297297 =11/37  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.187928 0.792453 10  
## 2 max f2 0.187928 0.905172 10  
## 3 max f0point5 0.187928 0.704698 10  
## 4 max accuracy 0.187928 0.702703 10  
## 5 max precision 0.998246 1.000000 0  
## 6 max recall 0.187928 1.000000 10  
## 7 max specificity 0.998246 1.000000 0  
## 8 max absolute\_mcc 0.187928 0.452856 10  
## 9 max min\_per\_class\_accuracy 0.627600 0.333333 9  
## 10 max mean\_per\_class\_accuracy 0.187928 0.656250 10  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_4)



**Rectifier with Dropout:**

nn\_5 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_5",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "RectifierWithDropout",  
 hidden = c(500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T  
)

h2o.performance(nn\_5,test.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.193127  
## RMSE: 0.4394622  
## LogLoss: 2.171004  
## Mean Per-Class Error: 0.3125  
## AUC: 0.7261905  
## Gini: 0.452381  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 6 10 0.625000 =10/16  
## 211 0 21 0.000000 =0/21  
## Totals 6 31 0.270270 =10/37  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.586526 0.807692 7  
## 2 max f2 0.586526 0.913043 7  
## 3 max f0point5 0.586526 0.724138 7  
## 4 max accuracy 0.586526 0.729730 7  
## 5 max precision 1.000000 0.833333 3  
## 6 max recall 0.586526 1.000000 7  
## 7 max specificity 1.000000 0.937500 0  
## 8 max absolute\_mcc 0.586526 0.504016 7  
## 9 max min\_per\_class\_accuracy 0.586526 0.375000 7  
## 10 max mean\_per\_class\_accuracy 0.586526 0.687500 7  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_5)



**Maxout with Dropout:**

nn\_6 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_6",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "MaxoutWithDropout",  
 hidden = c(500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T  
)

h2o.performance(nn\_6,test.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.4449376  
## RMSE: 0.6670364  
## LogLoss: 2.888669  
## Mean Per-Class Error: 0.3988095  
## AUC: 0.610119  
## Gini: 0.2202381  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 4 12 0.750000 =12/16  
## 211 1 20 0.047619 =1/21  
## Totals 5 32 0.351351 =13/37  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.019517 0.754717 10  
## 2 max f2 0.000000 0.867769 15  
## 3 max f0point5 0.019517 0.671141 10  
## 4 max accuracy 0.019517 0.648649 10  
## 5 max precision 1.000000 1.000000 0  
## 6 max recall 0.000000 1.000000 15  
## 7 max specificity 1.000000 1.000000 0  
## 8 max absolute\_mcc 0.999996 0.303895 3  
## 9 max min\_per\_class\_accuracy 0.998602 0.285714 7  
## 10 max mean\_per\_class\_accuracy 0.019517 0.601190 10  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_6)



h2o.performance(nn\_1,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1625316  
## RMSE: 0.4031521  
## LogLoss: 0.8764427  
## Mean Per-Class Error: 0.2624625  
## AUC: 0.8447447  
## Gini: 0.6894895  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 46 44 0.488889 =44/90  
## 211 4 107 0.036036 =4/111  
## Totals 50 151 0.238806 =48/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.023067 0.816794 44  
## 2 max f2 0.000234 0.908333 49  
## 3 max f0point5 0.542199 0.795107 41  
## 4 max accuracy 0.416495 0.761194 42  
## 5 max precision 0.542199 0.962963 41  
## 6 max recall 0.000000 1.000000 72  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.000234 0.548460 49  
## 9 max min\_per\_class\_accuracy 0.416495 0.522222 42  
## 10 max mean\_per\_class\_accuracy 0.416495 0.738589 42  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_2,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.2387207  
## RMSE: 0.4885905  
## LogLoss: 0.9372256  
## Mean Per-Class Error: 0.2711712  
## AUC: 0.8333333  
## Gini: 0.6666667  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 42 48 0.533333 =48/90  
## 211 1 110 0.009009 =1/111  
## Totals 43 158 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.120827 0.817844 59  
## 2 max f2 0.120827 0.913621 59  
## 3 max f0point5 0.862533 0.796460 52  
## 4 max accuracy 0.120827 0.756219 59  
## 5 max precision 0.985880 0.959184 44  
## 6 max recall 0.000173 1.000000 85  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.120827 0.554959 59  
## 9 max min\_per\_class\_accuracy 0.551938 0.504505 56  
## 10 max mean\_per\_class\_accuracy 0.120827 0.728829 59  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_3,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1475263  
## RMSE: 0.3840915  
## LogLoss: 0.658837  
## Mean Per-Class Error: 0.2701201  
## AUC: 0.8379379  
## Gini: 0.6758759  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 43 47 0.522222 =47/90  
## 211 2 109 0.018018 =2/111  
## Totals 45 156 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.566689 0.816479 61  
## 2 max f2 0.566689 0.908333 61  
## 3 max f0point5 0.954867 0.805970 55  
## 4 max accuracy 0.566689 0.756219 61  
## 5 max precision 0.999615 0.981132 52  
## 6 max recall 0.000000 1.000000 89  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.566689 0.548460 61  
## 9 max min\_per\_class\_accuracy 0.903940 0.495495 57  
## 10 max mean\_per\_class\_accuracy 0.954867 0.732132 55  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_4,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.2196784  
## RMSE: 0.4686986  
## LogLoss: 0.6327696  
## Mean Per-Class Error: 0.2711712  
## AUC: 0.8318318  
## Gini: 0.6636637  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 42 48 0.533333 =48/90  
## 211 1 110 0.009009 =1/111  
## Totals 43 158 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.155375 0.817844 63  
## 2 max f2 0.155375 0.913621 63  
## 3 max f0point5 0.746435 0.785498 54  
## 4 max accuracy 0.155375 0.756219 63  
## 5 max precision 0.998246 1.000000 0  
## 6 max recall 0.009154 1.000000 84  
## 7 max specificity 0.998246 1.000000 0  
## 8 max absolute\_mcc 0.155375 0.554959 63  
## 9 max min\_per\_class\_accuracy 0.226987 0.495495 61  
## 10 max mean\_per\_class\_accuracy 0.155375 0.728829 63  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_5,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1436784  
## RMSE: 0.3790493  
## LogLoss: 0.7531213  
## Mean Per-Class Error: 0.2656156  
## AUC: 0.8436436  
## Gini: 0.6872873  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 43 47 0.522222 =47/90  
## 211 1 110 0.009009 =1/111  
## Totals 44 157 0.238806 =48/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.084465 0.820896 46  
## 2 max f2 0.084465 0.915141 46  
## 3 max f0point5 0.956244 0.801749 41  
## 4 max accuracy 0.586526 0.761194 44  
## 5 max precision 1.000000 0.960000 8  
## 6 max recall 0.005610 1.000000 52  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.084465 0.563722 46  
## 9 max min\_per\_class\_accuracy 0.956244 0.495495 41  
## 10 max mean\_per\_class\_accuracy 0.586526 0.735435 44  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_6,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.2847582  
## RMSE: 0.5336274  
## LogLoss: 1.411247  
## Mean Per-Class Error: 0.2746246  
## AUC: 0.832032  
## Gini: 0.6640641  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 43 47 0.522222 =47/90  
## 211 3 108 0.027027 =3/111  
## Totals 46 155 0.248756 =50/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.019517 0.812030 59  
## 2 max f2 0.019517 0.901503 59  
## 3 max f0point5 0.514351 0.805970 54  
## 4 max accuracy 0.019517 0.751244 59  
## 5 max precision 1.000000 1.000000 0  
## 6 max recall 0.000000 1.000000 104  
## 7 max specificity 1.000000 1.000000 0  
## 8 max absolute\_mcc 0.019517 0.533549 59  
## 9 max min\_per\_class\_accuracy 0.514351 0.486486 54  
## 10 max mean\_per\_class\_accuracy 0.514351 0.732132 54  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

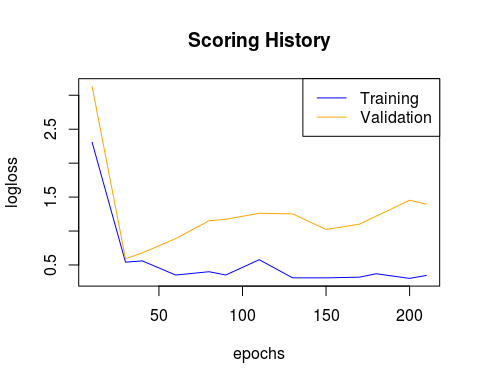
**Recitfier with Dropout with 2 additional hidden layers of 500 neurons each:**

nn\_7 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_7",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "RectifierWithDropout",  
 hidden = c(500,500,500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T,  
 l1=1e-06  
)

h2o.performance(nn\_7,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1582569  
## RMSE: 0.3978151  
## LogLoss: 0.9577213  
## Mean Per-Class Error: 0.2711712  
## AUC: 0.8334334  
## Gini: 0.6668669  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 42 48 0.533333 =48/90  
## 211 1 110 0.009009 =1/111  
## Totals 43 158 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.695753 0.817844 51  
## 2 max f2 0.695753 0.913621 51  
## 3 max f0point5 1.000000 0.796460 44  
## 4 max accuracy 0.695753 0.756219 51  
## 5 max precision 1.000000 0.951220 28  
## 6 max recall 0.000113 1.000000 60  
## 7 max specificity 1.000000 0.977778 0  
## 8 max absolute\_mcc 0.695753 0.554959 51  
## 9 max min\_per\_class\_accuracy 0.934974 0.504505 48  
## 10 max mean\_per\_class\_accuracy 0.695753 0.728829 51  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_7)



**Ensemble:**

Add the predictions from all 7 neural networks as features to dataset:

p1 <- as.data.frame(h2o.predict(nn\_1,dat.hex))

p2 <- as.data.frame(h2o.predict(nn\_2,dat.hex))

p3 <- as.data.frame(h2o.predict(nn\_3,dat.hex))

p4 <- as.data.frame(h2o.predict(nn\_4,dat.hex))

p5 <- as.data.frame(h2o.predict(nn\_5,dat.hex))

p6 <- as.data.frame(h2o.predict(nn\_6,dat.hex))

p7 <- as.data.frame(h2o.predict(nn\_7,dat.hex))

p1 <- p1[,3]  
p2 <- p2[,3]  
p3 <- p3[,3]  
p4 <- p4[,3]  
p5 <- p5[,3]  
p6 <- p6[,3]  
p7 <- p7[,3]  
  
p <- data.frame(cbind(p1,p2,p3,p4,p5,p6,p7))  
  
p <- scale(p)  
  
my.dat <- data.frame(cbind(my.dat,p))

h2o.shutdown(prompt=F)

## [1] TRUE

**K-means clustering**

Perform k-means clustering, with k=10, and add cluster group as a variable:

rm(list=ls())  
load("mydat.RData")  
k <- kmeans(x=my.dat[,-1],centers=10)  
k <- k$cluster  
k <- as.factor(k)  
  
require(dummies)  
  
k <- dummy(k)  
  
my.dat <- data.frame(cbind(my.dat,k))  
  
my.dat$k1 <- as.factor(my.dat$k1)  
my.dat$k2 <- as.factor(my.dat$k2)  
my.dat$k3 <- as.factor(my.dat$k3)  
my.dat$k4 <- as.factor(my.dat$k4)  
my.dat$k5 <- as.factor(my.dat$k5)  
my.dat$k6 <- as.factor(my.dat$k6)  
my.dat$k7 <- as.factor(my.dat$k7)  
my.dat$k8 <- as.factor(my.dat$k8)  
my.dat$k9 <- as.factor(my.dat$k9)  
my.dat$k10 <- as.factor(my.dat$k10)  
save(my.dat,file="mydat.RData")

rm(list=ls())  
load("mydat.RData")

**Support Vector Machines**

Predict using various kernels and add as features:

Linear Kernel:

require(e1071)  
sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="linear")  
svm.p <- data.frame(predict(sv,my.dat))

Polynomial Kernel:

sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="polynomial")  
svm.p <- data.frame(cbind(svm.p,predict(sv,my.dat)))

Radial Kernel

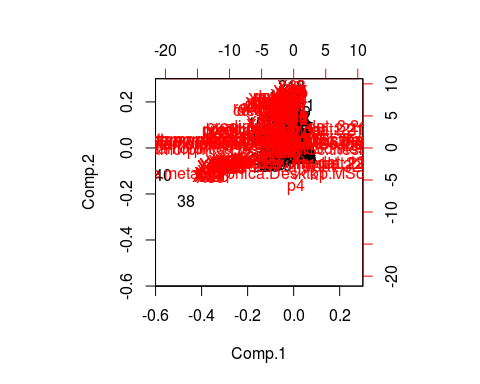
require(e1071)  
sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="radial")  
svm.p <- data.frame(cbind(svm.p,predict(sv,my.dat)))

Sigmoid Kernel:

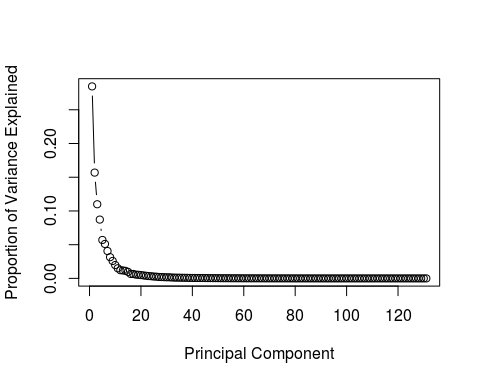
require(e1071)  
sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="sigmoid")  
svm.p <- data.frame(cbind(svm.p,predict(sv,my.dat)))

require(dummies)  
svm.p <- as.data.frame(svm.p)  
svm.p <- dummy.data.frame(svm.p)  
my.dat <- data.frame(cbind(my.dat,svm.p))

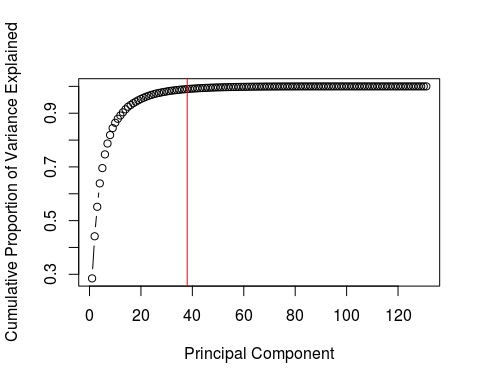
my.dat$k1 <- as.numeric(my.dat$k1)  
my.dat$k2 <- as.numeric(my.dat$k2)  
my.dat$k3 <- as.numeric(my.dat$k3)  
my.dat$k4 <- as.numeric(my.dat$k4)  
my.dat$k5 <- as.numeric(my.dat$k5)  
my.dat$k6 <- as.numeric(my.dat$k6)  
my.dat$k7 <- as.numeric(my.dat$k7)  
my.dat$k8 <- as.numeric(my.dat$k8)  
my.dat$k9 <- as.numeric(my.dat$k9)  
my.dat$k10 <- as.numeric(my.dat$k10)  
  
pc <- princomp(my.dat[,-1])  
biplot(pc)



std\_dev <- pc$sdev  
  
pr\_var <- std\_dev^2  
  
prop\_varex <- pr\_var/sum(pr\_var)  
  
plot(prop\_varex, xlab = "Principal Component",  
 ylab = "Proportion of Variance Explained",  
 type = "b")



plot(cumsum(prop\_varex), xlab = "Principal Component",  
 ylab = "Cumulative Proportion of Variance Explained",  
 type = "b")  
  
abline(v=min(which(cumsum(prop\_varex)>=.99)),col="red")



min(which(cumsum(prop\_varex)>=.99))

## [1] 38

new.dat <- data.frame(cbind(as.character(my.dat$pdgCode),pc$scores[,1:38]))  
new.dat$V1 <- as.factor(new.dat$V1)  
names(new.dat)[1] <- "pdgCode"  
  
for(i in 2:ncol(new.dat)){  
 new.dat[,i] <- as.numeric(as.character(new.dat[,i]))  
}

**Final prediction**

Build a random forest that uses all the new features, as well as the original dataset to classify particles as pdgcode 211 or pdgcode -211:

require(randomForest)  
  
rf <- randomForest(x=new.dat[1:100,2:39],y=new.dat$pdgCode[1:100])  
  
pp <- predict(rf,newdata = new.dat)  
  
pp <- data.frame(cbind(as.factor(new.dat$pdgCode),as.factor(pp)))  
pp[,3] <- pp[,1]==pp[,2]  
  
length(which(pp[,3])==T)/nrow(pp)

## [1] 0.7562189

1. At this stage, the ReLU model is essentially nothing more than a Multi-variable linear model: [↑](#footnote-ref-1)