# Neural networks for Arca

## Master Project

Maarten Post

June 2018

KM3NET outing

# Table of contents

# Introduction into neural networks

# A network of neurons

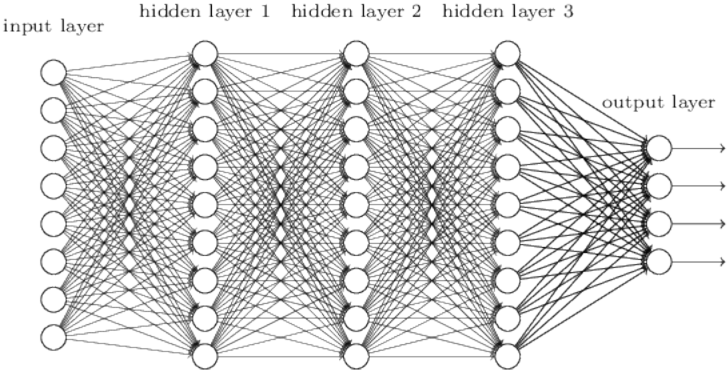



**Figure 1:** The common way to represent a neural network

# Mathematical notation

$$f(W\vec{x} + \vec{B})$$

With

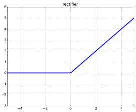
$x$ input data vector

$W$ weight matrix

$B$ bias vector

$f$ a non-linear "activation" function like tanh or $\frac{1}{1+e^{-x}}$, that often

$f : \mathbb{R} \rightarrow (-1, 1)$



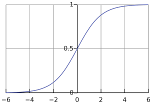



**Figure 2:** Relu- and the Sigmoid function

# Building a "network"

$$f(W_2 f(W_1 f(W_0 x + B_0) + B_1) + B_2) = y$$

If $x$ has n data point $W_0$ is a n $*$ m matrix and $B_0$ is a m vector.
$W_1$ is an m $*$ z matrix and $B_1$ a z vector, ect.
$f$ can be a different function each layer.
$y$ is the output vector

# Outputs

For classification one-hot encoding is common. Some activation functions normalise the output.

$$\hat{y} = \begin{bmatrix} \nu_e \\ \nu_\mu \\ {}^{40}K \\ \vdots \\ \mu_{atmos} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad y = \begin{bmatrix} 0.11 \\ .8 \\ 0 \\ \vdots \\ .09 \end{bmatrix}$$

Where $\hat{y}$ is the true label and $y$ the output of the neural network. The label with the highest output is chosen.

# Outputs

For regression / fitting the activation function of the last layer is left out so the output is continue. It is possible to predict one ore more values. For example.

$$\hat{y} = \begin{bmatrix} E \\ dx \\ dy \\ dz \end{bmatrix} = \begin{bmatrix} 10GeV \\ 0 \\ 0 \\ -1 \end{bmatrix} \qquad y = \begin{bmatrix} 8GeV \\ .1 \\ .1 \\ .8 \end{bmatrix}$$

# Cost function

The outputs are compared with each other with a cost function. This cost should be minimised by training. Common cost functions are square error- and cross entropy function.

$$C(y, \hat{y}) = \sum_i (y_i - \hat{y}_i)^2 \qquad C(y, \hat{y}) = -\sum_i \hat{y}_i \log(y_i)$$

# Training

Update the weights proportional to the gradient of the cost function.

$$\vec{W} = \vec{W} - \alpha \nabla C(\vec{W})$$

Where $\nabla C$ are the derivatives of C with respect to all matrix elements of the weights and biases. This is calculated with the chain-rule and means that all the activation functions should have a derivative.

# Computer vision

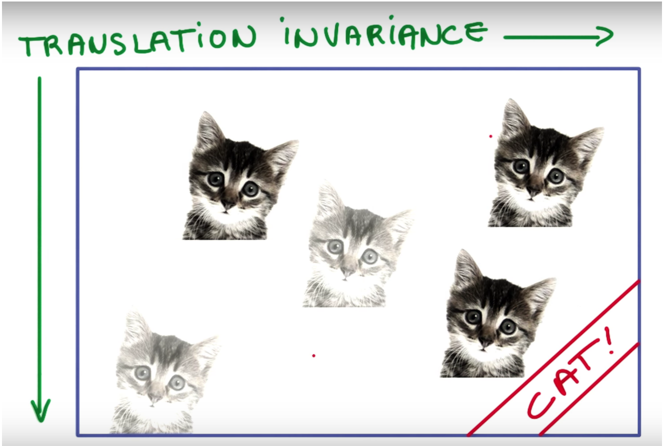



**Figure 3:** A Cat on different places in the picture. 1978 by 1330 pixels, total of $2.6 \times 10^6$ pixels

Too much weights needed to be computational efficient.
Picture shifted by one pixel can give total different output.

Convolutional networks are introduced to solve this problem and to take advantage of translation symetry.

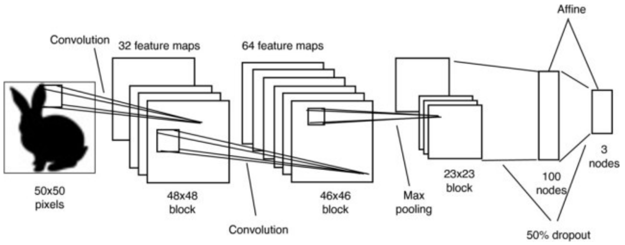



**Figure 4:** Convolutional neural network for classifing animals on pictures

# Convolution

$$(\text{Filter} * \text{Picture with rabbit}) = \text{new picture}$$

$$\left( \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 & \dots \\ 4 & 5 & 6 & \\ 7 & 8 & 9 & \\ \vdots & & & \ddots \end{bmatrix} \right) [1,1]$$

$$=$$

$$(i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9) + B$$

# Max pooling

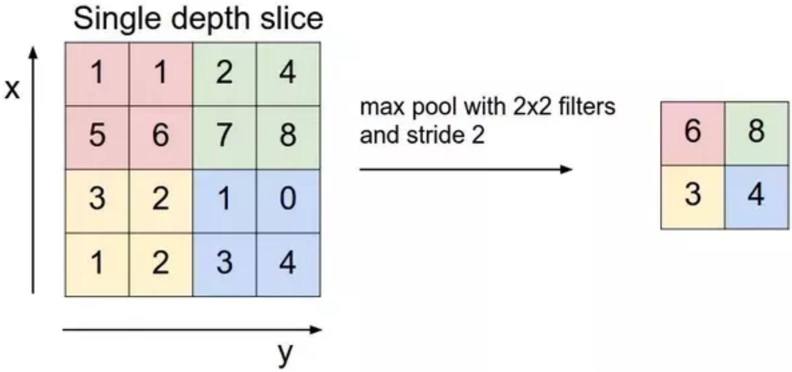



**Figure 5:** The max pooling of a matrix

# From pictures to particle physics

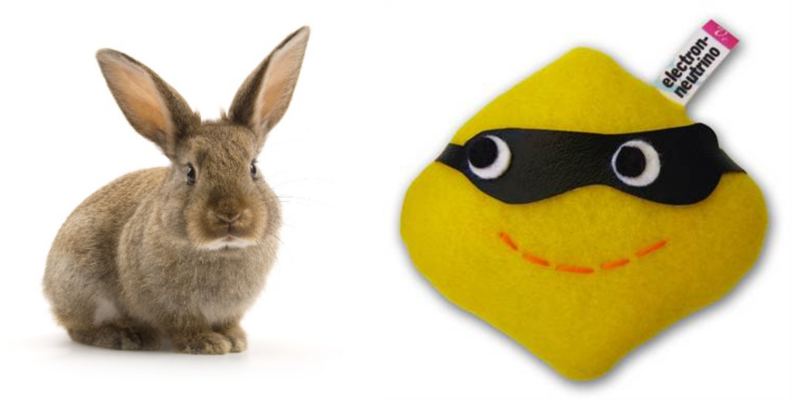



**Figure 6:** Rabbit and a electron neutrino

# Classification between showers, tracks and background

# Arca

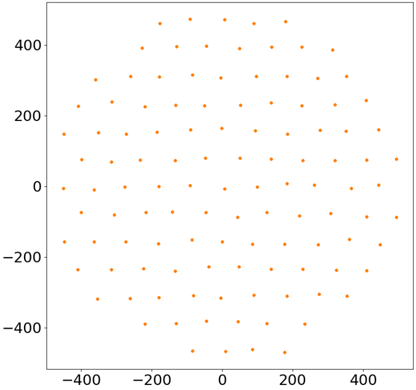



**Figure 7:** Top view of arca. meters

# Arca

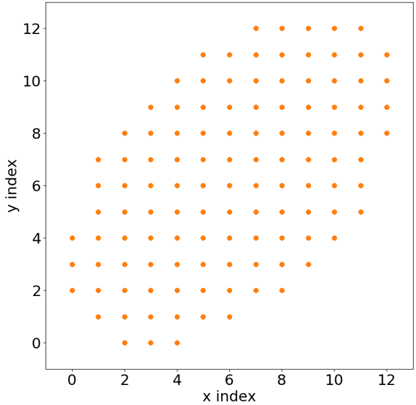



**Figure 8:** Matrix representation of Arca. Top view. index

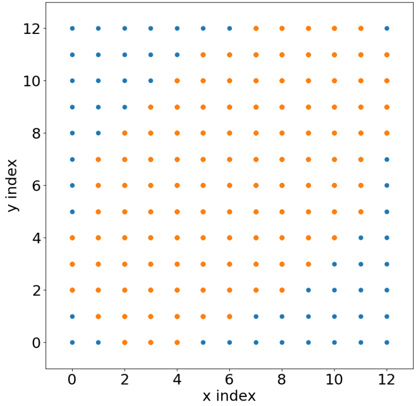



**Figure 9:** Matrix representation of Arca with padding. Top view. index

# Events

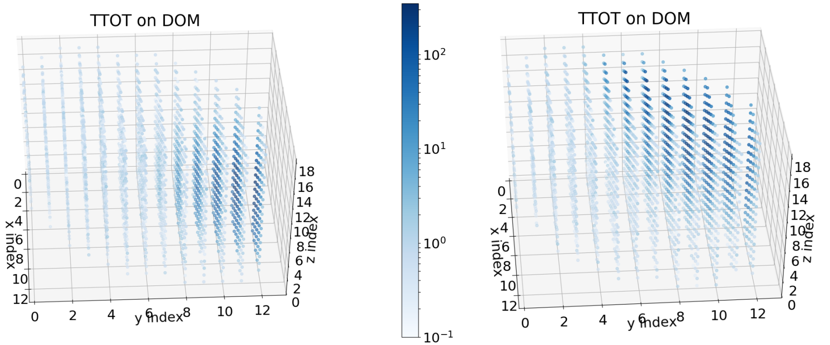



**Figure 10:** High energy shower and track ($5 \times 10^6$ GeV). Time integrated over $\sim$12000 ns

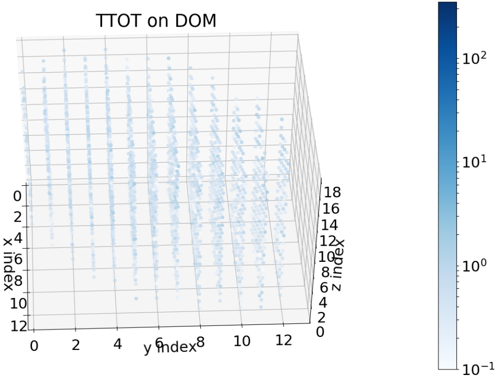



**Figure 11:** Only K40. Time integrated over 12000 ns

# KM3NNET

input 13, 13, 18, 1 event matrix

**Layer 1**

35 filters of 6, 6, 6

Activation function Relu and max pooling

output 7, 7, 9, 35

**Layer 2**

60 filters of 3, 3, 3

Relu and max pooling

output 4, 4, 5, 60

**Layer 3**

15 filters of 2, 2, 2

Relu and max pooling

output 2, 2, 3, 15

reshape to 180 vector

**Layer 4**

Normal neural network layer of 180 by 1028 matrix
Activation function Sigmoid

**Layer 5**

Normal neural network layer of 1028 by 60 matrix
Sigmoid

**Layer 6**

Normal neural network layer of 60 by 3 matrix
Activation function Softmax

output 3 vector "probability distribution for the classes"
Costfunction is cross entropy

# Confusion matrix

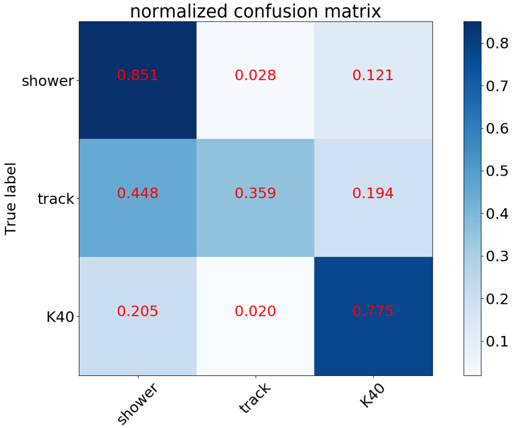



**Figure 12:** Confusion matrix

# Energy

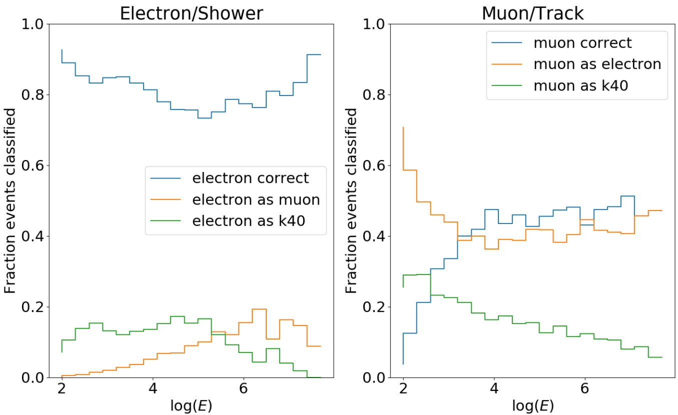



**Figure 13:** Energy distribution of classified events

# Number of hits

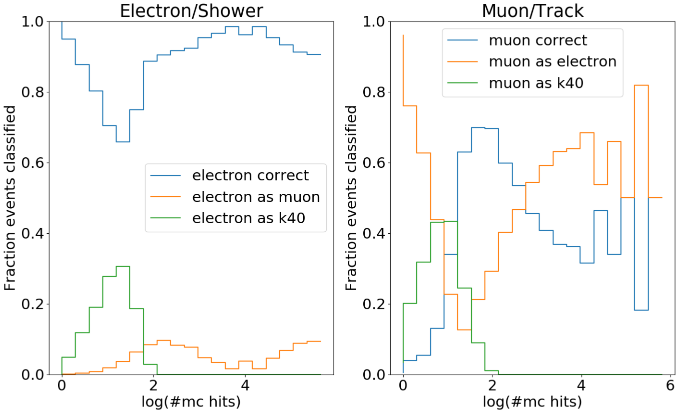



**Figure 14:** Number of monte carlo hits distribution of classified events

# Next step: time

# LSTM Network

LSTM Cell stands for Long short term memory cell and are often used for sequential data like text but can also be used for pictures. Output becomes input.

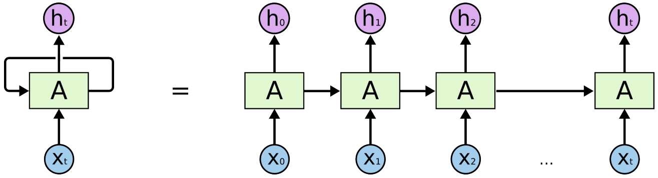



**Figure 15:** Simple LSTM network

Next time the mathematics, let's use them first.

# Time slices

**Layer 7**

Break events up in time slices and use each slide as input for KM3NNET. Then take the outputs as inputs for a LSTM network.

Shower

Track

K40