Particle Tracking with Deep Neural Networks at PANDA

Adeel Akram

Uppsala University
adeel.akram@physics.uu.se

$\overline{\text{P}}$ANDA Tracking Workshop
GSI Darmstadt

September 18, 2018

Outline

- Conventional tracking approach
- Motivation for new methods
- Intro to Machine Learning
- Data Preprocessing
- Model Deployment
- Further Investigation
- Next Steps

Track Reconstruction

Common approach in High Energy Physics:

- **Track finding**
  - Input: Particle Hits
  - Algorithm: e.g. SttCellTrackFinder, Hough Transform etc.
  - Output: Track Candidates

- **Track fitting**
  - Input: Track Candidates
  - Algorithm: e.g. Riemann Fit, Helix Fit etc.
  - Output: Track Kinematics

Track Reconstruction

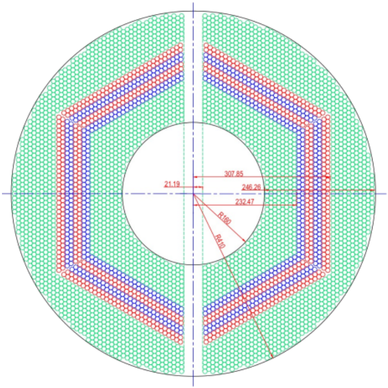
Conventional tracking methods suffer **one/more** of the following issues.

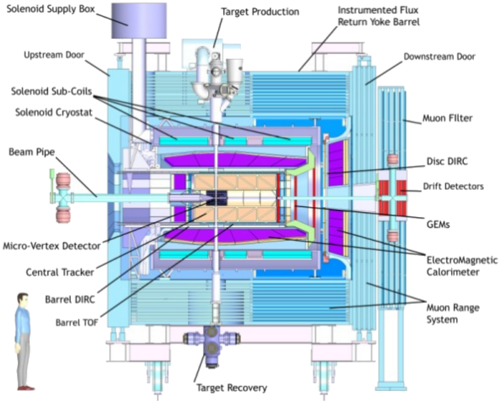
- Rely on linear dynamic models
- Are serial in nature
- Scale badly with track multiplicity
- Consume huge computing resources

An alternate approach is using Machine Learning (ML) methods.

- Track finding (pattern recognition)
- Track fitting (kinematics)

PANDA Experiment





Machine Learning

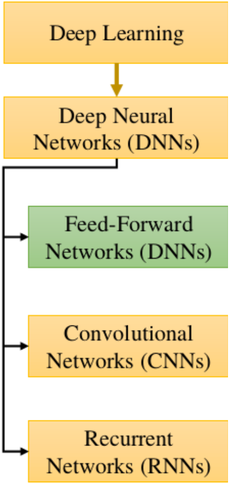
It is the ability of machines to learn complex representations of data through learning. Learning approaches are;

- **Supervised Learning**
  - Regression, Classification, **Patteren Recognition**
- Unsupervised Learning
  - Clustering, Density Measurements
- Reinforcement Learning
  - Robotics etc

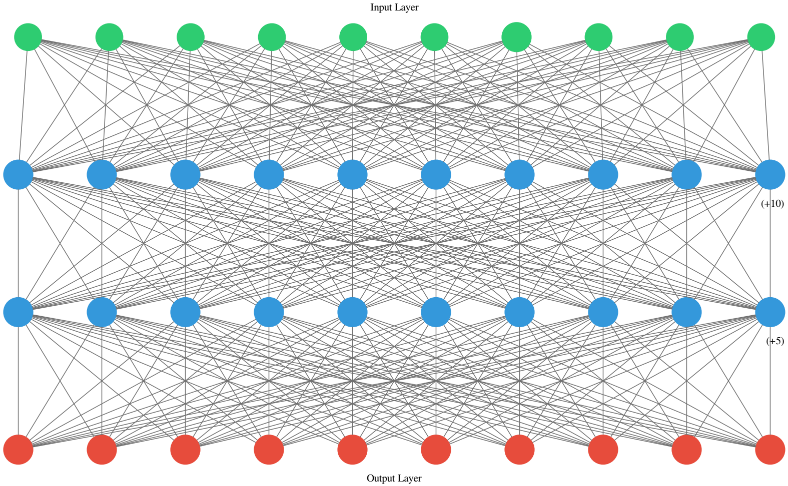
Deep Learning

Deep learning approach is to introduce multiple hidden layers in an existing model. Common network topologies are

- Feed-forward Neural Netowks (DNNs)
- Convolutional Neural Networks (CNNs or ConvNets)
- Recurrent Neural Networks (RNNs) and LTSM



Feed-forward Topology



Forward Propagation

Mathematics for Neural Nets:

$$z^{[l](i)} = W^{[l]}\, x^{(i)} + b^{[l]}$$
$$a^{[l](i)} = g^{[l]}(z^{[l](i)})$$
$$\hat{y}^{(i)} = a^{[L](i)}$$
$$l = 1, 2, ..., L$$
$$i = 1, 2, ..., m$$

l: l^{th} layer
i: i^{th} training example
x: input vector
b: bias vector
W: weight matrix
g: activation function
$\hat{y}$: estimate of final layer

Cost Function (J):

$$J(\hat{y}, y) = -1/m \sum_{i}^{m} [\hat{y}^{(i)} \log(y^{(i)}) + (1 - \hat{y}^{(i)}) \log(1 - y^{(i)})]$$

Back Propagation

After each epoch ($\equiv$ 1 execution cycle), error is back propagated using method of **gradient descent**. First,

$$dW^{[l]} = \frac{\partial J}{\partial W^{[l]}}$$
$$db^{[l]} = \frac{\partial J}{\partial b^{[l]}}$$

Second, update parameters with learning rate 'α':

$$W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha\, db^{[l]}$$

Prediction

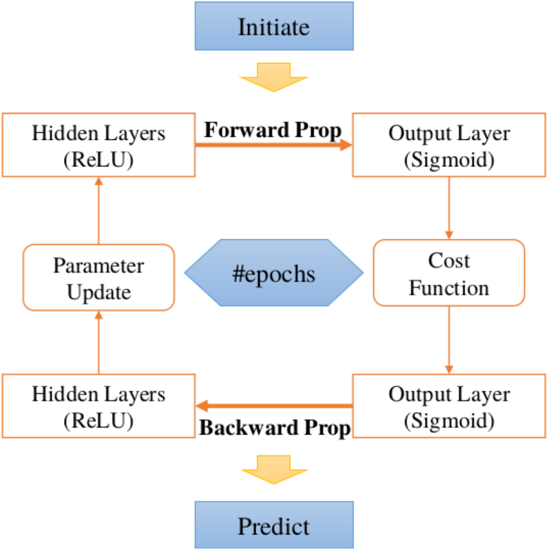
Whole execution of a Neural Network is reduced to minimization of **Cost Function (J)**.

$$J(\hat{y}, y) \rightarrow 0 \quad \text{given} \quad \alpha, W, b$$

After all epochs, our model is ready to predict using a threshold (τ).

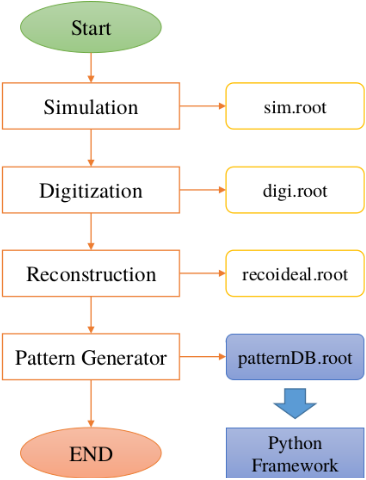
$$y^{(i)}_{prediction} = \begin{cases} 1, & \hat{y}^{(i)} > \tau \\ 0, & \text{otherwise} \end{cases}$$

Flow Diagram



Data Generation

- Data is generated using PandaROOT
- $p\bar{p} \rightarrow \Lambda\bar{\Lambda} \rightarrow \bar{p}\,\pi^+ + p\,\pi^-$ channel with $P_{beam} = 7.0$ GeV
- $nEvent = 10,000$ using "PndEvtGenDirect" generator
- Reconstruction is done using the "idealtracker"
- Output patterns (fired tubes) are then generated for proton tracks.



Pattern Generator

- A pattern generator class developed in Uppsala.
- Currently, avaiable in PandaROOT.
- This class is used to extract tube ids associated with a proton track.
- Finally, we should have Input/Output patterns (list of fired tubes)

Data Import

- Numpy Arrays for Vectorization
- Numpy Arrays are a requirement for Keras (TensorFlow) framework too.
- Data is imported into Python using uproot (minimalist ROOT I/O) from Scikit-HEP.
- uproot give access to ".root" data as Numpy Arrays.

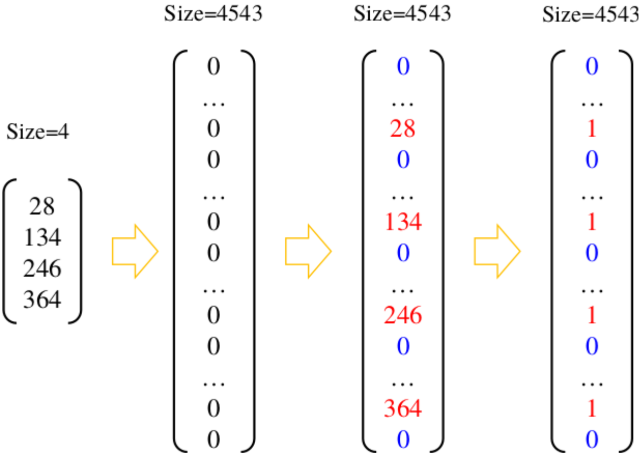
Input/Output Patterns

Imported data as pandas DataFrame (STTHitArray isn't available yet):

sr	Input (X)	Output (y)			
	patterns	patterns	p_x	p_y	p_z
1	[STTHitArray]	[31 101 104 ... 4493]	0.09237	0.10075	0.07186
2	[STTHitArray]	[19 22 102 ... 3648]	-0.0895	0.00896	0.02246
3	[STTHitArray]	[80 188 302 ... 4173]	0.23895	0.05811	0.32304
...	...	...	...	...	...

Our output patterns have diffrent sizes (i.e. # of fired tubes), not suitable for ANNs. Need patterns of fixed size (~ 4543).

Resizing Output Patterns



Train/Test Datasets

After resizing, we have input and output patterns with same fixed size.

As a final step data is divided into train_set (80%) and test_set (20%) using the **Scikit-learn** package for machine learning.

As STTHitArray isn't available in Python, so input patterns are replaced with output patterns for a self test.

Model Deployemnet

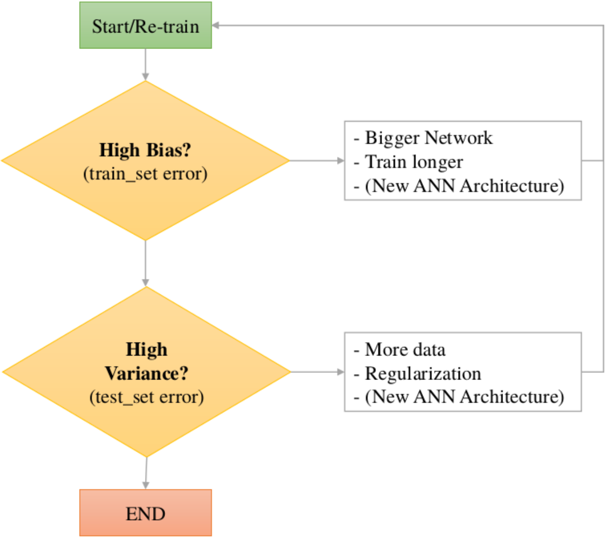
```
#!/usr/bin/env python3
def init_dff():
    model=Sequential()
    model.add(Dense(units=5000, activation='relu', kernel_initializer='glorot_uniform',
    input_dim=ip_dim))
    model.add(Dense(units=5000, activation='relu', kernel_initializer='glorot_uniform'))
    model.add(Dense(units=5200, activation='relu', kernel_initializer='glorot_uniform'))
    model.add(Dense(units=4000, activation='relu', kernel_initializer='glorot_uniform'))
    model.add(Dense(units=ip_dim, activation='sigmoid'))
    return model

model=init_dff()
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Execute:
history = model.fit(X_train, y_train, batch_size=128, epochs=1000, verbose=1,
          validation_data=(X_test, y_test))

# Predict:
y_pred = model.predict(X_test, verbose=1, batch_size=128)
```

Characterizing ANNs



Final Remarks

- Data Generation, Data Import and Data Preprocessing.
- A DNN is programmed in Keras (TensorFlow) and executed.
- Self-test shows code is "bug-free" and can be reused.
- Python and PandaROOT are bridged together using uproot.

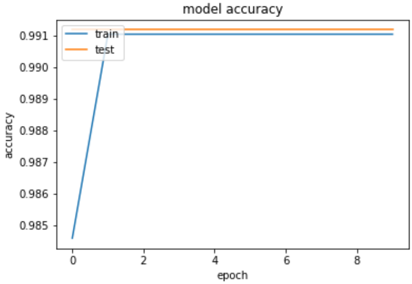
Next Steps

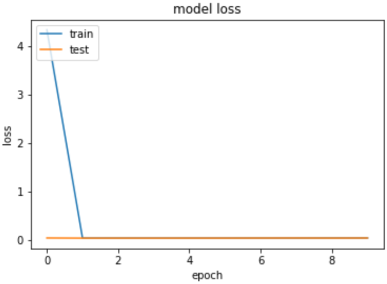
- Proper formulation of physics problem is needed (e.g. inclusion of Momentum Regression, PID etc).
- Complete run when **STTHitArray** is available
- Hyperparameter Tuning

Questions?

Backup Slides

Initial Run (Not Accurate)





Xavier Initialization

Weights and biases are intialized using samples from a truncated normal distribution centered on $\mu = 0$ with standard deviation:

$$\sigma = \frac{2}{n_{[l]} + n_{[l+1]}}$$

Where, n denotes the number of nodes in a layer l. Here, $n_{[l]}$ and $n_{[l+1]}$ are the dimensions of weight matrix (W) for layers $l, l+1$.

$$W \in \mathbb{R}^{n^{[l+1]} \times n^{[l]}}$$

Optimization

We used **Mini-batch** Gradient Decent (GD) in *Keras/TensorFlow* (also Batch & Stochastic GD). In addition, GD is optimized using **Adam** optimizer to escape saddle points where GD fails due zero gradient. Apart from Adam, we can also use

- Momentum
- RMSprop
- AdaGrad
- AMSGrad etc.

Under/Over fitting

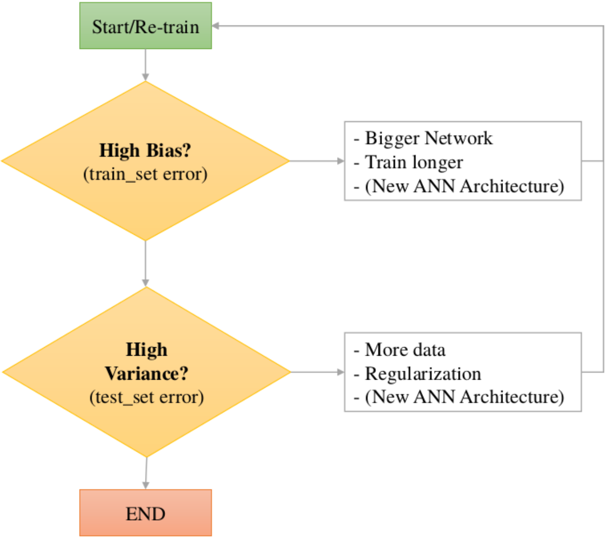
Errors in train/test datasets, it can tell whether our model is under/over (high bias/high variance) fitting our data.

- High Bias $\rightarrow$ under-fitting
- High Variance $\rightarrow$ over-fitting

How to get a Neural Network Model which is "just" right for our problem?

- Ans: Tune the Network.

Recipe for Investigation



Improving Neural Networks

Normalization:

- Input Data

Hyperparameters:

- W, b, α, λ etc.
- nodes & layers

Regularization:

- *L1 or L2*
- Dropout

Optimization:

- Adam

Regularization

In case our model overfits data (i.e. high variance), we can fix this by introducing a regularization term (known as L2) in our cost function to minimize this effect.

$$J(\hat{y}, y) = 1/m \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \left\| W^{[l]} \right\|_F^2 + \frac{\lambda}{2m} b^2$$

Where, λ is the regularization parameter, index F denotes $Frobenius$ norm. Other regularization method such as $L1$ and Dropout (turning off certain nodes, reducing the model dimensions) also exists.

Programming Framworks

C++ Environment:

- Standard C++/ROOT
- TMVA (Toolkit for Multivariate Analysis)
- etc.

Python Environment:

- Python 3.0
- Numpy for Vectorization
- TensorFlow/scikit-learn
- etc.

TrackML Competition 2018

Featured Prediction Competition

TrackML Particle Tracking Challenge

High Energy Physics particle tracking in CERN detectors

$25,000
Prize Money

CERN · 300 teams · 3 months to go (2 months to go until merger deadline)

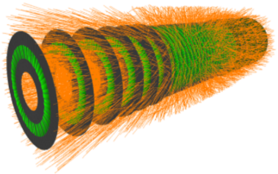
Overview | Data | Kernels | Discussion | Leaderboard | Rules | Team | My Submissions | **Submit Predictions**

Overview

- **Description**
- **Evaluation**
- **Prizes**
- **About The Sponsors**
- **Timeline**

To explore what our universe is made of, scientists at CERN are colliding protons, essentially recreating mini big bangs, and meticulously observing these collisions with intricate silicon detectors.

While orchestrating the collisions and observations is already a massive scientific accomplishment, analyzing the enormous amounts of data produced from the experiments is becoming an overwhelming challenge.



Additional Resources