Autonomous Fault Detection Using Artificial Intelligence Applied to CLAS12 Drift Chamber Data

August 16, 2018

A Bachelor’s Thesis by Christian Peters
Motivation

> Most crucial elements of a physical experiment?
  > Methods of measurement, e.g. drift chamber at CLAS12
  > Need to be highly precise
  > Essential for success

> Problem: Extreme conditions often lead to faults
  > Distortions in measurement accuracy
  > Have to be detected and filtered out during runtime

> Too much data to be processed by a human
  > An *autonomous* approach of fault detection is required
Motivation

> An emerging field lending itself particularly well to the task:
  > The domain of Artificial Intelligence (AI)
  > Deep Learning, Convolutional Neural Networks (CNNs)
> Goal: Apply methods of AI to the problem of fault detection
  > Experimental context: CLAS12 drift chamber
> Baseline software: deeplearning4j (DL4J) library
  > Will be used to implement the fault detection system
The CLAS12 Drift Chamber

> Subsystem of the CLAS12 particle detector
  > Electron beam hits target inside the detector’s center
  > Drift Chamber (DC) is used to measure the results (particle tracks)

> Hierarchical arrangement of multiple wires grouped together as wire chambers
  > Wires are used to detect particle presence
  > Particle hits wire $\rightarrow$ wire gets activated
The CLAS12 Drift Chamber

Figure: The hierarchical structure of a single wire chamber.
Drift Chamber Faults

> Drift chamber operates under extreme conditions
  > Huge amounts of radiation
  > Components can get damaged during an experiment
  > Single wires or collections thereof stop working

> Wire activations of a superlayer can be visualized as heatmaps
  > Easier to detect faults
Dead Wire
Dead Pin
Dead Connector
Dead Fuse
Hot Wire
Artificial Neural Networks

Figure: A common ANN-structure represented by a directed graph.
Modeling Artificial Neurons

Figure: The components of a single artificial neuron $k$. 
Activation Functions

- Determine the “activity”-level of a neuron based on the summed and weighted inputs
- Non-Linear
  - Enables the network to model complex relations
  - Multiple linear functions collapse into just a single linear function
Sigmoid Activation Function

\[ \phi(z) = \frac{1}{1 + e^{-\theta z}} \]  

- Transforms an input into a range between 0 and 1
- \( \theta \) adjusts the sensitivity with respect to the input
- Reduces the impact of outliers
- Often used in the early days
  - Biological inspiration, can also be interpreted as a “firing-rate”
Sigmoid Activation Function

Figure: The sigmoid activation function plotted for different values of $\theta$. 
Problems with the Sigmoid Activation Function

- We sometimes want to keep big values
  - Small values tend to fade out in deep networks (many hidden layers)
- “Saturates” for very big or negative inputs, i.e. does not change much when the input changes
  - This leads to training problems as we shall see later
ReLU Activation Function

\[ \phi(z) = \max(0, z) \]  

- Remedies the problems of the sigmoid function
- Cuts away negative values → sparsity among the neuron activations
  - Promotes simpler representations
- Actually more biologically inspired than the sigmoid
- Very easy to compute
ReLU Activation Function

Figure: The ReLU activation function.
Softmax Activation Function

\[
\phi(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}
\]  

> Usually applied to the output neurons
> Outputs can be interpreted as probabilities
  > Useful in classification, every possible class gets a probability
> Using the exponential function before normalization amplifies bigger signals and attenuates weaker ones
  > Helpful in training
> Interpretation of the \( z_i \): Unnormalized log-probabilities
Neural Networks as Classifiers

> We successfully established a mathematical model of neural networks

> How can we train them to perform classification tasks?
    > Remember, we want to classify what kinds of faults are in a superlayer within the drift chamber

> To do this, let’s first take a look at classification in general
Classification

> The data consists of attributes as well as class labels
> Goal: Predict the class label by only looking at the attributes
> First step: Training
  > The classification algorithm (classifier) is presented with many training examples
  > For every new example, the classifier adjusts its parameters to improve its classification ability
  > This is done to build a predictive model
> Second step: Testing
  > Some new testing examples are presented to the classifier that it did not see during training
  > These examples are used to determine, if the classifier learned any useful concepts from the training data, i.e. to generalize
Evaluating a Classifier

> The results of the testing phase are entered into a confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>Class Positive (Predicted)</th>
<th>Class Negative (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Positive (Actual)</td>
<td>True Positives (TP)</td>
<td>False Negatives (FN)</td>
</tr>
<tr>
<td>Class Negative (Actual)</td>
<td>False Positives (FP)</td>
<td>True Negatives (TN)</td>
</tr>
</tbody>
</table>

> This matrix is used to compute evaluation metrics like accuracy.
Training the Network

Which parameters can be adjusted during training?

- The weights and biases store the network’s knowledge and need to be tuned to improve performance.
- Other parameters like number of layers or activation function are set in advance (hyperparameters).

How to adjust the weights and biases?

- Measure the error on a batch of training examples.
- Minimize the error by taking a step of gradient descent ($\nabla error$).
- Repeat this for a number of passes through the training data (one pass = one epoch).

After training, test the network on new examples.

- Compute evaluation metrics.
- Was it able to generalize?
Convoluotional Neural Networks

> Simple ANNs work well for moderate amounts of attributes
  > Problems arise when amount of inputs grows
  > Number of parameters (weights and biases) “explodes”
  > Requires huge amounts of space and nearly impossible to train

> Sometimes, the input has a specific structure
  > E.g. images are arranged in grids of pixels (fault heatmaps are similar)
  > Every pixel has a local relevance
  > No need to connect every neuron to every input

> Use that structure to create simpler models that are easier to train
Convolution Layers

- Arrange the neurons in a grid, just like the input
- Every neuron “watches” a specific area, the *local receptive field*
  - Weights are shared → less parameters
- Works just like a sliding window (similar to a *convolution*)
- Multiple convolutions are performed → stack of hidden grids
Pooling Layers

> Reduce the input’s complexity by downsampling
  > Every neuron just remembers the maximum of its local receptive field
> Forget about the exact location of a feature
  > Leads to spatial invariance

![Diagram of pooling operation]

The Convolutional Architecture

> Stack multiple convolution and pooling layers
  > These are used to extract relevant features
> Use a fully connected layer in the end to perform classification
> The network is also trained via gradient descent
  > Weights and biases are updated in each step to minimize classification error
Implementing the Fault Detector

> Build a convolutional neural network in DL4J
  > Easy to monitor training and compute evaluation metrics
  > Fast due to C++ backend engine

> First, data has to be normalized
  > Activation levels can vary across superlayers
  > We only care about the distinct fault patterns
  > Scale wire activations from 0 to 1

> Many architectures and parameters were tried
  > Network too shallow $\rightarrow$ unable to learn complex faults (e.g. two dead wires next to each other)
  > Multiple faults per superlayer are possible $\rightarrow$ multiple networks were trained, each specializing in a single fault
Final Network Architecture

- **Raw Data of Superlayer**
  - Normalization
  - **Input Layer 6x112**
  - **Convolution 40@2x3**
    - ReLU
  - **Convolution 30@2x2**
    - ReLU
  - **Pooling 2x2**
    - ReLU
  - **Convolution 20@2x2**
    - ReLU
  - **Fully Connected Layer, 100 Neurons**
    - ReLU
  - **Softmax Output**
    - Neuron 1: Fault
    - Neuron 2: No Fault
Training the Fault Detector

- Used Michaels simulation suite
  - Based on real world background signals
  - Randomly inserts fault combinations and generates class labels accordingly
- Each classifier was trained on 100,000 examples
- Testing was done using 10,000 new examples from the simulator
  - Accuracy > 97%
Real Data Validation

- Need to show that the detector not only works on simulated data
  - Did it extract some general concepts?
- Tested the system on some real world examples to show its strengths and weaknesses
> Faults display a sharp contrast → classifier works well
  > Dead pin and dead wire classifier both report 100% fault
  > The other classifiers don’t detect their fault with 99% certainty
Blurred fault $\rightarrow$ classifier struggles

- Classifier reports 99% no fault for the pin
- We believe that more real data can solve this
Two Dead Wires

> Classifier detects two dead wires next to each other
> Reports 93.29% certainty for the wires and 100% for the pin
Conclusion

> Convolutional Neural Networks work well for fault detection
> Blurred fault problem will be solved in the future
  > Will use real world blurred faults during training
  > After all, a deep learning system can only be as good as the data it was trained on
> Next step: fault localization
  > Need to know, where exactly a fault is located
  > State-of-the-art: YOLOv3, a CNN specialized in object localization
  > Use the present system as a pre-stage classifier
> Excited to see, how the system will perform on the hundreds of petabytes of real CLAS12 drift chamber data


Artificial Neural Networks

- Class of machine learning algorithms
  - Loosely inspired by biological nervous systems
- Collection of artificial neurons that are connected with each other
  - Enables them to exchange signals along their connections
  - Can be represented by a directed graph
- Usually arranged in layers
  - *Input Layer* collects input signals and passes them on
  - *Hidden Layers* apply transformations to incoming signals and pass the outcomes further into the network
  - *Output Layer* applies a final transformation representing the networks’ result
- Goal: Convert input into meaningful output by applying multiple transformations
Components of the neural model

> A set of weighted inputs
  > Each input originating from neuron \( j \) and traveling into neuron \( k \) is first multiplied by a weight \( w_{kj} \)

> A summation unit
  > All the weighted inputs are summed and a constant value, the \( bias \), is added to yield the result \( z_k \)

> An activation function
  > Applies a non-linear transformation \( \phi(\cdot) \) to the output of the summation unit
  > This result, called \( y_k \), is propagated further into the network alongside the connections
The Role of the Bias Value

> The bias is added as a constant to the sum of the weighted inputs in the summation unit
> Acts like a threshold that has to be overcome
  > Negative bias: Positive weighted inputs needed for the neuron to become active
  > Positive bias: Negative weighted inputs needed to stop the neuron from being active
The Role of the Bias Value

Figure: The sigmoid activation function plotted for different bias values.
Evaluation Metrics

> **Accuracy:**
  > Percentage of testing examples that were classified correctly

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} 
\]

(4)

> **Precision:**
  > Percentage of correctly classified examples among all examples classified as positive

\[
\text{Precision} = \frac{TP}{TP + FP} 
\]

(5)
Evaluation Metrics

> Recall:
  > What percentage of positive examples was classified correctly?

\[
Recall = \frac{TP}{TP + FN}
\]  

(6)

> F1 Score:
  > Harmonic mean of precision and recall

\[
F1 \text{ Score} = \frac{2 \times Precision \times Recall}{Precision + Recall}
\]  

(7)
Dead Wire Classifier

- Accuracy: 97.61%
- Precision: 99.96%
- Recall: 95.65%
- F-Measure: 97.76%

<table>
<thead>
<tr>
<th></th>
<th>Dead Wire (Predicted)</th>
<th>No Dead Wire (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Wire (Actual)</td>
<td>5212</td>
<td>237</td>
</tr>
<tr>
<td>No Dead Wire (Actual)</td>
<td>2</td>
<td>4549</td>
</tr>
</tbody>
</table>

Table: Confusion matrix of the dead wire classifier.
Dead Pin Classifier

- Accuracy: 99.95%
- Precision: 99.92%
- Recall: 99.98%
- F-Measure: 99.95%

<table>
<thead>
<tr>
<th></th>
<th>Dead Pin (Predicted)</th>
<th>No Dead Pin (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Pin</td>
<td>4739</td>
<td>1</td>
</tr>
<tr>
<td>(Actual)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Dead Pin</td>
<td>4</td>
<td>5256</td>
</tr>
<tr>
<td>(Actual)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table: Confusion matrix of the dead pin classifier.
Dead Connector Classifier

> Accuracy: 98.77%
> Precision: 99.23%
> Recall: 95.69%
> F-Measure: 97.43%

<table>
<thead>
<tr>
<th>Dead Connector (Actual)</th>
<th>Dead Connector (Predicted)</th>
<th>No Dead Connector (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Connector</td>
<td>2334</td>
<td>105</td>
</tr>
<tr>
<td>(Actual)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Dead Connector</td>
<td>18</td>
<td>7543</td>
</tr>
<tr>
<td>(Actual)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table: Confusion matrix of the dead connector classifier.
Dead Fuse Classifier

- Accuracy: 98.95%
- Precision: 97.32%
- Recall: 98.20%
- F-Measure: 97.76%

<table>
<thead>
<tr>
<th></th>
<th>Dead Fuse (Predicted)</th>
<th>No Dead Fuse (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Fuse (Actual)</td>
<td>2288</td>
<td>42</td>
</tr>
<tr>
<td>No Dead Fuse (Actual)</td>
<td>63</td>
<td>7607</td>
</tr>
</tbody>
</table>

Table: Confusion matrix of the dead fuse classifier.
Dead Channel Classifier

> Accuracy: 99.11%
> Precision: 98.84%
> Recall: 98.64%
> F-Measure: 98.74%

<table>
<thead>
<tr>
<th></th>
<th>Dead Channel (Predicted)</th>
<th>No Dead Channel (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Channel (Actual)</td>
<td>3493</td>
<td>48</td>
</tr>
<tr>
<td>No Dead Channel (Actual)</td>
<td>41</td>
<td>6418</td>
</tr>
</tbody>
</table>

Table: Confusion matrix of the dead channel classifier.
Hot Wire Classifier

- Accuracy: 100.00%
- Precision: 100.00%
- Recall: 100.00%
- F-Measure: 100.00%

<table>
<thead>
<tr>
<th></th>
<th>Hot Wire (Predicted)</th>
<th>No Hot Wire (Predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot Wire (Actual)</td>
<td>5532</td>
<td>0</td>
</tr>
<tr>
<td>No Hot Wire</td>
<td>0</td>
<td>4468</td>
</tr>
</tbody>
</table>

Table: Confusion matrix of the hot wire classifier.