Applying deep learning to FRB classification

Liam Connor
16 February 2017
Swinburne FRB
Apertif PAF increases FoV by ~30
Search pipeline tested

- Dozens of Crab GPs detected by pipeline
- Single pulses from B0329+54
- Clean band
TensorFlow

- Opensource software developed by Google Brain for internal use
- Builds / runs computational graph representing NNs
- Provides ML visualisation toolkit “TensorBoard”
- Easily run on CPUs, GPUs, or TPUs
- Makes use of pre-existing highly-optimised numerical libraries
Training set: true-positives

- Inject simulated bursts into real data
- Draw from broad distribution of width, scattering, scintillation, etc.
Training set: false-positives

- Use real false positives
- RFI / dropped packets / thermal triggers too hard to simulate
Dynamic spectrum CNN
Feature extraction (1d-convolution + pooling + dropout)

false positive
FRB

Feature extraction (2d-convolution + pooling + dropout)

false positive
FRB

Feature extraction (2d-convolution + pooling + dropout)

false positive
FRB

Feature extraction (2d-convolution + pooling + dropout)

false positive
FRB
Feature extraction (2d-convolution + pooling + dropout)

Feature extraction (1d-convolution + pooling + dropout)

Feature extraction (2d-convolution + pooling + dropout)

Merge DNNs to hybrid net

false positive
FRB
Real FRB
Merge DNNs to a hybrid network

Feature extraction (2d-convolution + pooling + dropout)

Feature extraction (1d-convolution + pooling + dropout)

Feature extraction (2d-convolution + pooling + dropout)

false positive

FRB
Real-time signal detection?
Real-time signal detection?
Real-time signal detection?
Real-time signal detection?
Real-time signal detection?
Convolution

\[ N \]

\[ \ast \]

\[ 1 \]

\[ \mathcal{O}(N^2) \]

\[ \mathcal{O}(N \log_2 N) \]
Convolution

\[ \begin{array}{c}
1 \\
N \\
N \\
\end{array} \quad \begin{array}{c}
\ast \\
\ast \\
\ast \\
\end{array} \quad \begin{array}{c}
N \\
N \\
N \\
\end{array} \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Brute-Force Complexity</th>
<th>FFT Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolution</td>
<td>( \mathcal{O}(N^2) )</td>
<td>( \mathcal{O}(N \log_2 N) )</td>
</tr>
<tr>
<td></td>
<td>( \mathcal{O}(N^4) )</td>
<td>( \mathcal{O}(N^2 \log_2^2 N) )</td>
</tr>
</tbody>
</table>
Convolution

**Brute-force FFT**

- **O(N^2)**
- **O(N^4)**
- **O(MN \times m^2)**

**FFT**

- **O(N \log_2 N)**
- **O(N^2 \log_2^2 N)**
- **O(MN \log_2 M)**

*highly optimised for GPUs*

*overlap-add method*
Convolution

**brute-force**

\[ \mathcal{O}(N^2) \quad \mathcal{O}(N^4) \]

**FFT**

\[ \mathcal{O}(N \log_2 N) \quad \mathcal{O}(N^2 \log_2^2 N) \]

- **overlap-add method**
  - \[ \mathcal{O}(MN \times m^2) \quad \mathcal{O}(MN \log_2 M) \]
- **highly optimised for GPUs**
- **FFT**
  - \[ \mathcal{O}(MN \log_2 M) \]
  - **overlap-add method**

\[ \mathcal{O}(N^2) \quad \mathcal{O}(N \log_2 N) \]

\[ \mathcal{O}(N^4) \quad \mathcal{O}(N^2 \log_2^2 N) \]
Convolution: $\mathcal{O}(kN_tN_f \log_2(N_t))$
Convolution: \( \mathcal{O}(n_k N_t N_f \log_2(N_t)) \)

brute force \( \mathcal{O}(N_t N_f N_{dm}) \)

tree de-dispersion \( \mathcal{O}(N_t N_f \log_2 N_f) \)

DNN parameters
• Working code to train / apply a hierarchical DNN to FRB candidates

• High recall / accuracy can be attained with a few thousand labelled triggers

• Let me know if you’d like to try it!

• Real-time classification could be faster than traditional dedispersion, but not ideal for FRBs