Practicalities of Neural Nets

Daniel Lawson — University of Bristol

Lecture 09.2 (v1.0.1)

Signposting

- This Block is split into two Lectures:
  - 09.1 (this lecture) on the theory
  - 09.2 on practicalities
- This is Lecture 2.

ILOs

- ILO1 Be able to access and process cyber security data into a format suitable for mathematical reasoning
- ILO4 Be able to use high throughput computing infrastructure and understand appropriate algorithms
- ILO5 Be able to reason about and conceptually align problems involving real data to appropriate theoretical methods and available methodology to correctly make inferences and decisions

Implementations

- Implementations are best though of in two classes.
- **Simple networks** have a restricted architecture and can be deployed "out of the box" as a Machine Learning tool.
  - Examples include sklearn.linear_model.Perceptron, R's neutralnet packages, etc
  - Often either shallow or very simple hidden layer structure
- **Deep networks** require a complex specification of architecture and significant computational optimisation, so are very large (and mercifully, open source) endeavours
  - This is the focus here.

Depp NN Implementations

- There are two main libraries for deep neural networks:
- **TensorFlow**, developed by Google Brain.
  - Well documented
  - Easier to use
  - Industry standard
  - Tensorboard visualisation is useful
- **PyTorch**, developed by Facebook.
  - Newer, less support
  - Dynamical coding paradigm: graph can remodel in the light of the data
  - Debugging is easier? As the code is compiled at runtime, like native python

Using implementations

- Tensorflow is a low-level language. You can interact with it through abstraction layers which allows very simple implementations.
  - Keras is very widely used and makes accessing TensorFlow very easy.
  - PyTorch is already conceptually a “high level” implementation.
- Keras can use various **backends** (implementations):
  - TensorFlow
  - MXNet
  - Theano is a pure python library for a wide class of array computation, not just Neural Networks. It was forked into Aesara…
  - Microsoft Cognitive Toolkit, but this is no longer in active development.
- See Tensorflow or keras?

Practical advice

- Explore recommendations. e.g. Practical Advice for Building Deep Neural Networks:
- As a starting point:
  - Use the "adam" optimizer
  - Use a ReLU activation function
  - Remember not to use an activation function for the output layer (except for classification, when use a sigmoid)
  - Add bias to every layer (shouldn't have to worry about this in keras)
  - Whiten (normalize) your input data (we'll see this in the workshop)
- **Don't believe me.** Get other opinions, and try things yourself.

Debugging

- Check the **input data**…
- For many tasks:
  - **OVERFIT**. "Accuracy should be essentially 100% or 99.99%". If it isn't, the network isn't flexible enough, or learning correctly.
- Change the learning rate
- Decrease mini-batch size
- Remove batch normalization (this exposes NA values)
- Reconsider the architecture
- PLOT your results! training loss by epoch is a natural plot

Workshop

- jupyter notebook
- Basic Bluecrystal usage
- All ready for the assignment?

Signposting

- The next topic is parallel algorithms, to compare with fast single node approaches.
- By the end of the course, you should:
  - Understand the tools available for neural networks
  - Be able to use high-level implementations efficiently

Further reading

- Keras and PyTorch
- Tensorflow or keras?
- A performance focussed comparison: TensorFlow, PyTorch or MXNet?
- Tensorboard
- Brilliant.org on Backpropagation
- Practical Advice for Building Deep Neural Networks