Assignment 2 – Understanding the task

# Document 1 – Deepnet Iface

**Purpose**

Learn to solve classification tasks

Hands-on with Tensorflow

Learn to build interface for Tensorflow, so that network architectures can be tested on a diverse array of dataset

**What to do**

Use Tensorflow to classify data from different sources.

**Things to note**

Number of layers

The datasets for this assignment should not require extremely deep networks, but anywhere from one to three hidden layers may work optimally, dependent upon your choice of the other parameters.

Activation and error functions to experiment with (All is TF primitives)

Activation functions:

* Sigmoid
* Hyperbolic tangent (tanh)
* Rectified linear units (RELUs)

Error functions (TF primitives):

* Mean-square error (MSE)
* Cross-entropy

Learn the mapping between the datasets and their best architecture

**Goal**

A primary goal of this assignment is to streamline this trial-and-error process so that many different networks can be experimented with in a short period of time.

Suggestion (by me(Jonas)) for **GUI**:  
<http://www.cosc.canterbury.ac.nz/greg.ewing/python_gui/>

**Demonstration:**

Run your system on any of the datasets from deepnet\_iface\_details.pdf

This process must occur without the need to re-compile or otherwise re-make your system. Your system must easily process the scenario specifications for each new run and display the appropriate graphics (such as weight arrays, dendrograms, etc.).

During a performance test:

* You get a dataset
* You choose an appropriate architecture and parameter setting
* Explain the behavior of the ANN arch. On a self chosen dataset(not bit-counter)
  + Use all of the following graphic tools
    - Mappings
    - Weight-and-bias visualizations
    - Dendrogram
  + Generate all of them before presentation

**Grade based on**

* Designofinterface to Tensorflow
  + Easy config and run neural nets with varying architectures
  + Possibility to choose data
  + Visualization of result
* Design appropriate ANN architecture for particular tasks
* The performance of these architectures
* Explanation of the behavior by visualizing:
  + Layer activation patterns
  + Weight matrices
  + Other relationships’ (as dendrograms)

# Document 2 – Deepnet Iface Details

**Intro**

This document includes description of concepts, techniques and tools.

Building a system that serves as an interface.

System must accept user-specification of:

* Characteristics of NN
* Data set
* Training / testing regime

**Requirements for system**

* Must be object oriented
* Same modularity as GANN and GANNmodule in tutor3.py
* Can build from scratch or by using tutor3.py

**Network scenario**

Consists of:

1. Dataset – **what to expect**
   1. Consists of cases which has a set of features along with a class
2. Architecture - **what to choose**
   1. Nr of hidden layers
   2. Hidden activation function
   3. Output activation
   4. Cost function
   5. Learning rate
   6. Initial weight range
   7. Data source – one of:
      1. File
      2. Function (see tflowtools.py)
   8. Case fraction
   9. Validation fraction (VeF = [0,1])
      1. S\*VaF
   10. Test fraction (TeF = [0,1])
       1. S\*TeF
   11. Training fraction (1-(VaF+TeF))
       1. S\*(1 –(TeF+VaF))
   12. Minibatch size
   13. Map batch size
   14. Steps
   15. Map Layers
   16. Map dendrograms
   17. Display weights
   18. Display biases
   19. Vint(validation interval)
   20. UNSURE about these:
       1. Nodes in each layer
       2. Error function
       3. Initial weight values
3. Training and testing - **what to do**
   1. Separate into training, validation and testing
      1. Training = 80 % or more; Randomly selected (10%val, 10%test)
   2. Repeatedly training (nr of epochs)
   3. Intermittently run the validation cases while recording errors (no learning)
   4. When total error has reach acceptance; then turn of backprop and test
      1. Correctly classified training cases
      2. Correctly classified test cases (indicator of ability to generalize)
4. Visualization – **what to present (Tensorboards for process visualized)**  
   Users most be able to turn these on and of and to specify more details about them
   1. Plot of progression of training-set and validation-set error
      1. Must be on the same graph (detect overtraining)
      2. Each point is the average (per case) of the minibatch
      3. X = nr of minibatches, y = error
   2. Listing of error percentage for the training and test sets
   3. Display of weights and biases for any user-chosen area (at the end)
      1. Example weights between layer 1 and 2.
   4. Set of corresponding activation levels from post training mapping run ????
   5. Dendrograms

Note:

Mapping – small sample through network with learning off; Display activation levels of user chosen set of layers. For any layer, a comparison of the different activation vectors can then serve as the basis for a dendrogram.

Weight and Bias viewing – graphic views of the weighst.

**Chapter 3 shows generation of all data sets, as for example:**

* gen\_all\_parity\_cases
  + From tflowtools.py
  + Generates all bit vectors of a specified length
  + Tagged with 1=odd, 0=even
  + Reasonably good on 10 bit vectors with 80% training is **95%.**

**Chapter 4 focus on internal representation of NN**

* **Visualize mapping**
  + Do\_mapping should be equal to do\_testing in tutor3.py
* **Dendrograms explained**
  + Sim and diffs of NN activation pattern
  + RELU (no upper bound) vs Sigmoid(bounded by 0,1)
  + Tflowtools.py has function **dendrogram**
    - Handles all graphics given two lists

**Chapter 5 explains how a NN performs a task**

**API:**

* in\_top\_k(predictions,targets,k)
  + TF primitive error function
  + Use when testing instead of MSE or CE.
  + K should be 1