Basic

- There are 4 main parts in a layer program:
    - constants
    - data
    - networks
    - scripts

Constants

- There are three different constants to define (default values):
    - threads: number of threads for parallelization (4)
    - batch size: size of the batch for the network (100)
    - log: log file where some messages are saved (netparser.log)

```
const{
  threads=4
  batch=10
  log="mnist.log"
}
```

Data

- Data can be read in ascii or binary. Data format is the following:

> samples dim output
>
> sample1_1 ... sample1_dim out1_1 .. out1_c
>
> ...
>
> samplen_1 ... samplen_dim outn_1 .. outn_c

- *samples* is the number of samples (int)
- *dim* is the dimensionality of the samples (int)
- *out* is ths number of output targets, classes in a classification problem (int)

- each row is a sample (float values)

Note: when samples are color images the channels appear separated, firts R, then G and then B and *dim* must be $3 \times rows \times cols$

- An example of data definition:

```
data {
  D1 [filename="../training", binary]
  D2 [filename="../test", binary]
  Dval [filename="../valid", ascii]
}
```

- *D1*, *D2* and *Dval* are the name of the data variables
- Full path to file can be used

Networks

- Networks are the main part of a Layer program
- Networks are composed by three main parts:
  - Data
  - Layers
  - Connections

- The networks are defined like this:

```
network N1 {
    ...
}
```

- where *N1* is the name of the network

Networks - Data

- The Networks have to defined at least a training data set
- Test and validation data sets are optional:

```
network N1 {
  //data
  data tr D1  //mandatory
  data ts D2 //optional
  data va Dval //optional
  ...
}
```

- When test or validation data are provided the error function of the net will be also evaluated for that data sets

Networks - Layers

- There are different layers that can be created:
  - **FI**: Input Fully Connected layer
  - **CI**: Input Covolutional layer
  - **F**: Fully Connected layer
  - **FO**: Ouput layer
  - **C**: Convolutional layer
  - **MP**: MaxPooling layer
  - **CA**: Cat layer

Networks - Layers - FI

- FI has not parameters just serve as a connection with the data

```
//NETWORK
network N1 {
  //data
  data tr D1

  // Fully Connected Input
  FI in
  ...
}
```

- *in* is the name of the layer
- The number of units is the dimensionality representation of the data

Networks - Layers - CI

- Convolutional input has three mandatory parameters that indicate how the raw data has to be mapped into an input Map
- The parameters are:
  - nz: number of channels
  - nr: image rows
  - nc: image cols

```
//NETWORK
network N1 {
  //data
  data tr D1

  // Convolutional Input for MNIST
  CI in [nz=1, nr=28, nc=28]
  ...
}
```

- *in* is the name of the layer

Networks - Layers -CI

- Convolutional input has optional parameters:
  - cr: crop rows
  - cc: crop cols

- Images will be randomly cropped on training phase
- Images will be center-cropped in test phase

```
//NETWORK
network N1 {
  //data
  data tr D1

  // Convolutional Input for MNIST
  // Images will be randomly cropped to 24x24
  CI in [nz=1, nr=28, nc=28, cr=24, cc=24]
  ...
}
```

Networks - Layers - F

- A fully connected layer

```
//NETWORK
network N1 {
  //data
  ...

  // Fully connected
  F  f1 [numnodes=1024]
  ...
}
```

- *f1* is the name of the layer
- *numnodes* is a mandatory parameter

Networks - Layers - FO

- FO has a mandatory parameter *{classification,regression}* that define the cost error: cross-entropy or mse respectivelly
- For regression, optionally, we can define an *autoencoder*

```
network N1 {
  ...
  // Targets are the output in data (one-hot)
  FO  out1 [classification]  // Cross-entropy

  // Targets are the output in data (real values)
  FO  out2 [regression]  // mse over output

  // Targets are the same input in data
  FO  out3 [regression,autoencoder] // mse over input
 ...
}
```

Networks - Layers - C

- Convolutional layers have three mandatory parameters:
  - nk: number of kernels
  - kr: height of kernel
  - kc: width of kernel

```
network N1 {
  ...
  // Convolutional Layer
  C c1 [nk=32, kr=3, kc=3]
  ...
}
```

- *c1* is the name of the layer
- Convolutional layers have the following optional parameters:
  - rpad: indicates if padding is done in the rows (0)
  - cpad: indicates if padding is done in the cols (0)
  - stride: stride value (1)

```


network N1 {
  ...
  // Convolutional Layer with padding
  // To keep the same size of Input Maps
  // use rpad=cpad=1
  C c1 [nk=32, kr=3, kc=3,rpad=1,cpad=1]

  // Stride=2 and only padding in cols
  C c2 [nk=32, kr=3, kc=3,cpad=1,stride=2]
  ...
}
```

Networks - Layers - MP

- MaxPool layers have two mandatory parameters:
  - sizer: height of the pooling region
  - sizec: width of the pooling region

```
network N1 {
  ...
  // Maxpool layer sizer=sizec=2
  MP p1[sizer=2,sizec=2]
  ...
}
```

Networks - Layers - CA

- Cat layers does not require any parameter

```
network N1 {
  ...
  // CAT Layer
  CA cat
  ...
}
```

Networks - Connections

- To connect two layers use the symbols "->"

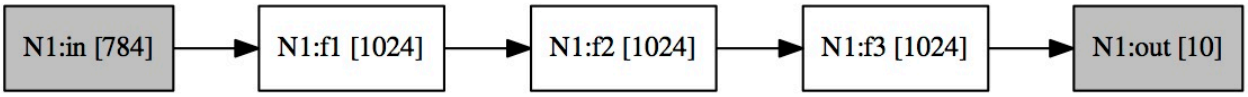
```
network N1 {
  ...
  c1->c2
}
```

Networks - Examples

Example 1 - MLP MNIST

```
// A typical MLP for MNIST
// 784->1024->1024->1024->10

//CONSTANTS
const{
  threads=8
  batch=10
  log="mnist.log"
}

// DATA FILES
data {
  D1 [filename="../training", binary]
  D2 [filename="../test", binary]
}

//NETWORK
network N1 {
  //data
  data tr D1  //mandatory
  data ts D2 //optional

  // Fully Connected Input
  FI in

  // Fully connected
  F  f1 [numnodes=1024]
  F  f2 [numnodes=1024]
  F  f3 [numnodes=1024]

  // Fully Connected Output
  FO  out [classification]

  // Links
  in->f1
  f1->f2
  f2->f3
  f3->out
}
```

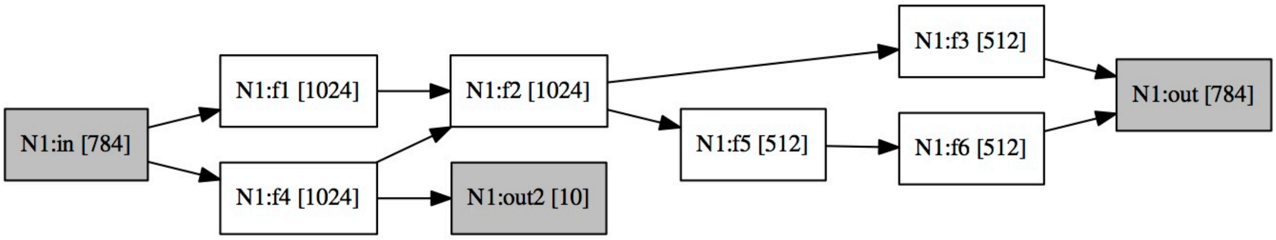
- layers generates a *dot* file with the name of each network
- Use " dot -T pdf N1.dot > N1.pdf" to generate a PDF with the network:



- but yes, more complicated networks can be defined

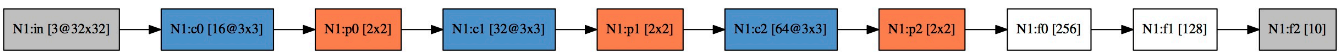
Example 2 - Hybrid MLP MNIST

```
//NETWORK
network N1 {
  //data
  data tr D1
  data ts D2

  // Fully Connected Input
  FI in

  // Fully connected
  F  f1 [numnodes=1024]
  F  f2 [numnodes=1024]
  F  f3 [numnodes=512]

  // Fully connected
  F  f4 [numnodes=1024]
  F  f5 [numnodes=512]
  F  f6 [numnodes=512]


  // Output
  FO  out [regression,autoencoder]
  FO  out2 [classification]

  // Connections
  in->f1
  f1->f2
  f2->f3
  f3->out

  in->f4
  f4->f2
  f4->out2

  f2->f5
  f5->f6
  f6->out

}
```



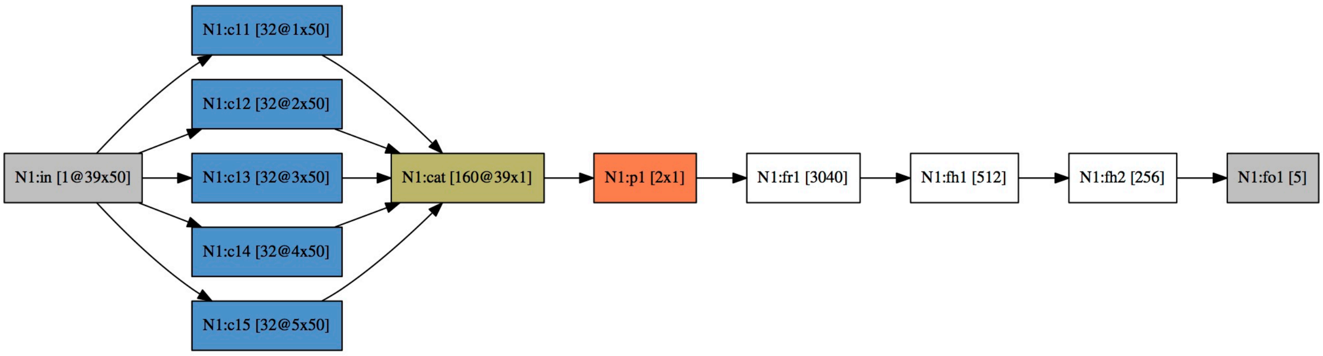
Example 3 - Convolutional CIFAR

```
network N1 {
  data tr D1
  data ts D2

  CI in [nz=3, nr=32, nc=32]

  C c0 [nk=16, kr=3, kc=3]
  MP p0[sizer=2,sizec=2]
  C c1 [nk=32, kr=3, kc=3]
  MP p1 [sizer=2,sizec=2]
  C c2 [nk=64, kr=3, kc=3]
  MP p2 [sizer=2,sizec=2]

  // FC reshape
  F   f0 []

  // FC hidden
  F  f1 [numnodes=128]

  // FC output
  FO f2 [classification]

  // Conecctions
  in->c0
  c0->p0
  p0->c1
  c1->p1
  p1->c2
  c2->p2
  //reshape
  p2->f0
  f0->f1
  f1->f2

}
```



Example 4 - CAT layers

```
network N1 {
  data tr D1
  data ts D2

  CI in [nz=1, nr=39, nc=50]

  C c11 [nk=32, kr=1, kc=50,rpad=1]
  C c12 [nk=32, kr=2, kc=50,rpad=1]
  C c13 [nk=32, kr=3, kc=50,rpad=1]
  C c14 [nk=32, kr=4, kc=50,rpad=1]
  C c15 [nk=32, kr=5, kc=50,rpad=1]
  MP p1 [sizer=2,sizec=1]

  CA cat

  // FC reshape
  F   fr1 []

  // FC hidden
  F  fh1 [numnodes=512]
  F  fh2 [numnodes=256]

  // FC output
  FO fo1 [regression]

  // Connections
  in->c11
  in->c12
  in->c13
  in->c14
  in->c15

  c11->cat
  c12->cat
  c13->cat
  c14->cat
  c15->cat

  cat->p1

  p1->fr1
  fr1->fh1
  fh1->fh2
  fh2->fo1

}
```



Scripts

- Script is a block of actions. There are two different types of actions:
    - functions
    - parameters modifications

Scripts Functions

- There are several functions that can be applied to the defined objects:

    - For Data:
        - Zscore(): To normalize Data
        - yuv(): to convert RGB Maps to YUV maps

    - For Networks:
        - train(epochs): to train networks

> More functions will be soon implemented

```
script {

  D1.yuv()
  D2.yuv()

  D1.zscore()
  D2.zscore(D1) //normalize D2 with D1 statistics

  N1.train(100)
}
```

Scripts Parameters

- There are several parameters that can be modified:

  - mu: learning rate
  - mmu: momentum rate
  - l2: l2 regularization (weight decay)
  - l1: l1 regularization
  - maxn: maxnorm regularization
  - noiser: noise ratio after activation function
  - noisesd: standard deviation of noise ($N(0.0, \sigma)$)
  - drop: dropout (0<drop<1)
  - bn: batch normalization ({0,1})
  - act: activation (0 Linear, 1 Relu, 2 Sigmoid, 3 ELU)
  - flip: to flip input images ({1,0})
  - shift: to shift randomly input images

- Parameters can be modified for one particular layer:

```
script{
...
N1.c2.drop=0.5
N1.in.flip=1
N1.in.shift=2
...
}
```

- Or, parameters can be modified for all the layers of the same network:

```
script{
...
N1.mu=0.001
N1.l2=0.0005
// noise in all the layers of the network:
N1.noiser=0.5
N1.noiser=0.3
...
}
```

- Some examples of scripts:

Example 1

```
script {

  D1.yuv()
  D2.yuv()

  D1.zscore()
  D2.zscore(D1)

  N1.in.flip=1.0
  N1.in.noiser=0.5
  N1.in.noisesd=1.0

  N1.f1.drop=0.5

  N1.l2=0.0005

  N1.bn=1.0

  // Learning rate annealing every 15 epochs
  N1.mu=0.01
  N1.train(15)

  N1.mu=0.005
  N1.train(15)

  N1.mu=0.0025
  N1.train(15)
}
```

Example 2

```
script {

  D1.zscore()
  D2.zscore(D1)

  N1.noiser=0.0
  N1.noisesd=1.0
  N1.in.noiser=0.5
  N1.mu=0.001
  N1.maxn=3.0

  N1.train(100)

}
```

The whole picture

Example 1

```
// A typical MLP
// 784->1024->1024->1024->10
// Check N1.dot and get a pdf of the net with:
// dot -T pdf N1.dot > N1.pdf

//CONSTANTS
const{
  threads=8
  batch=10
  log="mnist.log"
}

// DATA FILES
data {
  D1 [filename="../training", binary]
  D2 [filename="../test", binary]
}

//NETWORK
network N1 {
  //data
  data tr D1  //mandatory
  data ts D2 //optional

  // Fully Connected Input
  FI in

  // Fully connected
  F  f1 [numnodes=1024]
  F  f2 [numnodes=1024]
  F  f3 [numnodes=1024]

  // Fully Connected Output
  FO  out [regression]

  // Links
  in->f1
  f1->f2
  f2->f3
  f3->out
}
```

```
//RUN SCRIPT
script {

  D1.zscore()
  D2.zscore(D1)


  //N1.noisesd=1.0
  //N1.noiser=0.5
  N1.mu=0.01
  //N1.maxn=3.0

  N1.train(10)

}
```

Example 2

```
const {
threads=4
batch=100
}
data {
  D1 [filename="training", binary]
  D2 [filename="test", binary]
}


network N1 {
  data tr D1
  data ts D2

  // Covolutional Input
  CI in [nz=3, nr=32, nc=32]

  C c0 [nk=16, kr=3, kc=3]
  MP p0[sizer=2,sizec=2]
  C c1 [nk=32, kr=3, kc=3]
  MP p1 [sizer=2,sizec=2]
  C c2 [nk=64, kr=3, kc=3]
  MP p2 [sizer=2,sizec=2]

  // FC reshape
  F   f0 []

  // FC hidden
  F  f1 [numnodes=128]
```

```
  // FC output
  FO f2 [classification]

  // Connections
  in->c0
  c0->p0
  p0->c1
  c1->p1
  p1->c2
  c2->p2
  //reshape
  p2->f0
  f0->f1
  f1->f2

}

// RUN
script {

  D1.zscore()
  D2.zscore(D1)

  N1.mu=0.1

  N1.in.noiser=0.5
  N1.in.noisesd=1.0
  //N1.bn=1.0

  N1.train(1000)
}
```

ADVANCED ISSUES

Networks sharing layers

TODO

Training several networks

TODO

...