A-54299
Library CWI

L4

Technische documentatie functionele deel Neural Vision software

Michiel van Wezel
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

april 1998

Table of Contents

CWI BIBLIOTHEEK

3 0054 00117 6891

1. Inleiding

Dit is de technische documentatie behorende bij het functionele deel van de Neural Vision 2.0 software. De software is geschreven in de taal C++ en bestaat uit een aantal classes. Deze classes kunnen worden onderverdeeld in ondersteunende classes en classes die specifieke functionaliteit voor Neural Vision implementeren, vanaf hier NV2-specifieke classes genoemd.

In de categorie 'ondersteunende classes en modules' vallen de volgende classes en modules

- `fvector` class voor vectoren van `floats`.
- `fmatrix` class voor matrices van `floats`.
- `intvector` class voor vectoren van `int`'s.
- `intmatrix` class voor matrices van `int`'s.
- `my_exception` class voor exception handling.
- `random` module voor random number utilities.

In de categorie 'NV2-specifieke classes' vallen de volgende classes

- `dataset` class voor inlezen dataset en initiële bewerking daarvan.
- `sanger` class voor Sanger neuraal netwerk.
- `ica` class voor ICA neuraal netwerk.
- `fscnn` class voor fscnn neuraal netwerk.

In secties 2 en 3 worden deze classes één voor één besproken. In sectie 4 wordt van de 'NV2-specifieke classes' classes uitgelegd hoe zij gebruikt dienen te worden. Als laatste wordt in sectie 5 een korte toelichting op de genomen ontwerpbeslissingen gegeven.

2. Ondersteunende classes

In deze sectie worden de ondersteunende classes besproken.

2.1 De `fvector` *class voor vectoren van* `floats`

De `fvector` class is geïmplementeerd in de bestanden `fvector.h` en `fvector.cpp`. Bij alle member functions geldt dat waar nodig een bounds-check wordt uitgevoerd. Wordt deze check niet doorstaan, dan wordt er een exception van het type `my_exception` gegenereerd. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables

`int size`: Aantal elementen in de fvector

`float *v`: Pointer naar `float` array die getallen bevat

Member functions

`fvector()`: Default constructor. Dient niet te worden gebruikt.

`fvector(int s)`: Argument bearing constructor. Zet `size = s`.

`float &operator[](int)`: Overloading van de indexing operator `[]`. Voor een `fvector a` levert `a[i]` het `i`-de element op van `a`.

`fvector &operator=(fvector&)`: Overloading van de assignment operator =. Voor `fvector a` en `fvector b` voert `a=b` een toekenning van de respectievelijke waarden van `b` aan de respectievelijke waarden van `a` uit.

`fvector &operator=(float)`: Overloading van de assignment operator =. Voor `fvector a` en `float f` voert `a=f` een toekenning van `f` aan alle waarden van `a` uit.

`void dump(char *s)`: Dumpt alle elementen van de `fvector` naar `stdout` vorafgegaan door string `s`.

`void dump(ofstream& ofs, char *s)`: Dumpt alle elementen van de `fvector` naar `ofs` vorafgegaan door string `s`.

`void dump(ofstream& ofs, char *s, int n)`: Dumpt de eerste `n` elementen van de `fvector` naar `ofs` vorafgegaan door string `s`.

`~fvector()`: Destructor.

Verder bevat de module waarin de `fvector` class is gedefiniëerd een functie `distance(fvector&,fvector&)` die de euclidische afstand tussen twee vectoren retourneert.

2.2 De `fmatrix` *class voor matrices van* `floats`

De `fmatrix` class is geïmplementeerd in de bestanden `fmatrix.h` en `fmatrix.cpp`. Bij alle member functions geldt dat waar nodig een bounds-check wordt uitgevoerd. Wordt deze check niet doorstaan, dan wordt er een exception van het type `my_exception` gegenereerd. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables

`int nof_rows`: Aantal rijen in de matrix.

`int nof_cols`: Aantal kolommen in de matrix.

`fvector **m`: Pointer naar een array van `fvector` pointers. Deze `fvectors` bevatten de eigenlijke data.

Member functions

`fmatrix()`: Default constructor. Dient niet te worden gebruikt.

`fmatrix(int r, int c)`: argument bearing constructor. Zet `nof_rows = r nof_cols = c`.

`int rows_equal(int r1, int r2)`: Boolean functie die 1 oplevert wanneer rijen `r1` en `r2` identiek zijn, 0 anders.

`void dump(char *s)`: Dumpt de `fmatrix` naar `stdout`, voorafgegaan door string `s`.

`void dump(ofstream ofs, char *s)`: Dumpt de `fmatrix` naar `ofs` voorafgegaan door string `s`.

`void bare_dump(ofstream& ofs)`: Dumpt de `fmatrix` naar `ofs`.

`void bare_dump(ofstream& ofs, int n)`: Dumpt de eerste `n` elementen van iedere rij naar `ofs`.

`void bare_dump_file(char *fn)`: Creëert een file met naam `fn` en dumpt hierin de `fmatrix`.

`void init()`: Initialiseert alle elementen van de `fmatrix` op 0.

`float row_dist(int a, int b)`: Retourneert de Euclidische afstand tussen rij `a` en rij `b`.

`float row_dist_first_n_elements(int a, int b, int n)`: Retourneert de Euclidische afstand tussen rij `a` en rij `b` waarbij alleen de eerste `n` elementen van de rijen gebruikt worden voor het berekenen van de afstand.

`fvector &operator[](int a)`: Overloading van de indexing operator `[]`. Voor `fmatrix m` levert `m[a]` de `fvector` op die de `a`-de rij van `m` voorstelt.

`fmatrix &operator=(fvector& v)`: Overloading van de assignment operator. Voor `fmatrix m` kopiëert `m=v` de waarden van `v` naar `m`. Merk op dat dit alleen mogelijk is wanneer het aantal kolommen van `m` gelijk is aan 1 en wanneer het aantal rijen van `m` gelijk is aan het aantal kolommen van `v`. Wanner niet aan deze voorwaarde is voldaan wordt `throw my_exception` gegenereerd.

`fmatrix &operator=(fmatrix& fm)`: Overloading van de assignment operator. Voor `fmatrix m` kopiëert `m=fm` de waarden uit `fmatrix fm` nar de respectievelijke posities in `fmatrix m`.

`fmatrix &operator=(float x)`: Overloading van de assignment operator. Voor `fmatrix m` initialiseert `m=x` alle waarden van `m` op `x`.

`fmatrix &operator+=(fmatrix& fm)`: Overloading van de `+=` operator. Voor `fmatrix m` telt `m += fm` de waarden van de elementen van `fm` op bij de respectievelijke elementen van `m`.

`fmatrix &operator-=(fmatrix& fm)`: Overloading van de `-=` operator. Voor `fmatrix m` trekt `m -=fm` de waarden van de elementen van `fm` af van de respectievelijke elementen van `m`.

`fmatrix &operator*=(float x)`: Overloading van de `*=` operator. Voor `fmatrix m` vermenigvuldigt `m*=x` alle elementen van `m` met de waarde `x`.

`~fmatrix()`: Destructor.

Verder bevat de module met de class-definition de volgende non-member functions:

`extern void fmatrix_mult(fmatrix&, fmatrix&, fmatrix&)`: Matrix multiplicatie. Het resultaat van de vermenigvuldiging van het tweede en derde argument wordt geretourneerd in het eerste argument.

`extern void fmatrix_transpose(fmatrix&, fmatrix&)`: Het eerste argument wordt de getransponeerde van het tweede argument.

`extern void assign_fmatrix_to_fvector(fvector&, fmatrix&)`: Kent elementen van een matrix met 1 kolom toe aan `fvector`.

2.3 De `intvector` *class voor vectoren van* `int`*'s*

De `intvector` class is geimplementeerd in de bestanden `INTVECTO.H` en `INTVECTO.CPP`. Bij alle member functions geldt dat waar nodig een bounds-check wordt uitgevoerd. Wordt deze check niet doorstaan, dan wordt er een exception van het type `my_exception` gegenereerd. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables

`int size`: Aantal elementen in de `intvector`.

`int *v`: Pointer naar `int` array die getallen bevat.

Member functions

`intvector()`: Default constructor. Dient niet te worden gebruikt.

`intvector(int s)`: Argument bearing constructor. Zet `size = s`.

`int &operator[](int)`: Overloading van de indexing operator `[]`. Voor een `intvector a` levert `a[i]` het `i`-de element op van `a`.

`intvector &operator=(int)`: Overloading van de assignment operator `=`. Voor `intvector a` en `int n` voert `a=n` een toekenning van `n` aan alle waarden van `a` uit.

`void dump(char *) s`: Dumpt alle elementen van de `intvector` naar `stdout` vorafgegaan door string `s`.

`void rand_perm()`: Permuteert de elementen van de `intvector` op een willekeurige wijze.

`~intvector()`: Destructor.

2.4 De `intmatrix` *class voor matrices van* `int`*'s*

De `intmatrix` class is geïmplementeerd in de bestanden `intmatrix.h` en `intmatrix.cpp`. Bij alle member functions geldt dat waar nodig een bounds-check wordt uitgevoerd. Wordt deze check niet doorstaan, dan wordt er een exception van het type `my_exception` gegenereerd. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables

`int nof_rows`: Aantal rijen in de matrix.

`int nof_cols`: Aantal kolommen in de matrix.

`intvector** m`: Pointer naar een array van `intvector` pointers. Deze `intvectors` bevatten de eigenlijke data.

Member functions

`intmatrix()`: Default constructor. Dient niet te worden gebruikt.

`intmatrix(int r, int c)`: Argument bearing constructor. Zet `nof_rows = r` en `nof_cols = c`.

`int rows_equal(int a, int b)`: Retourneert `1` als rijen `a` en `b` aan elkaar gelijk zijn, `0` anders.

`intvector &operator[](int a)`: Overloading van de indexing operator `[]`. Voor `intmatrix m` levert `m[a]` de `intvector` op die de `a`-de rij van `m` voorstelt.

`void dump(char *s)`: Dumpt de `intmatrix` naar `stdout`, voorafgegaan door string `s`.

`void init()`: Initialiseert alle elementen van de `intmatrix` op `0`.

`~intmatrix()`: Destructor function.

2.5 De `my_exception` *class voor exception handling*

De `my_exception` class is geïmplementeerd in de bestanden `my_exception.h` en `my_exception.cpp`. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables

geen

Member functions

`my_exception()`: Default constructor. Genereert exception met message ëxception not specified".

`my_exception(char *s)`: Argument bearing constructor. Genereert exception met string `s` als message.

`char* what()`: Drukt message van exception af op `stdout`.

2.6 De `random` *module voor random number utilities*

De `random` module is een reeds langer bestaande module. Hij is geschreven in de programmertaal C. Deze module bevat de volgende functies.

Functies

`unsigned int init_rand()`: Initialiseert de random generator op basis van de systeemtijd. Retourneert de `unsigned int` die als seed gebruikt is.

`void init_rand_seed(unsigned int seed)`: Initialiseert de random generator op `seed`.

`float rando(float) m`: Retourneert een random `float` tussen 0 en `m`.

`int flip_coin()`: Retourneert 0 met 50 % kans, 1 met 50 % kans.

`int hrand(int high)`: Retourneert een random `int` in het bereik `[0..high-1]`.

`float random_between_bounds(float l,float u)`: Retourneert een random `int` tussen `l` en `u`.

3. NV2-SPECIFIEKE CLASSES

In deze sectie worden de NV2-specifieke classes besproken.

3.1 De `dataset` *class voor inlezen dataset en initiële bewerking daarvan*

De `dataset` class is geïmplementeerd in de bestanden `dataset.h` en `dataset.cpp`. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables Voor deze member variables geldt dat zij worden gealloceerd en geïnitialiseerd door de constructor function.

`int nof_vars`: Het totale aantal variabelen in de dataset (numeriek + nominaal).

`int nof_nom_vars`: Het aantal nominale variabelen in de dataset.

`int nof_num_vars`: Het aantal numerieke variabelen in de dataset.

`int nof_patterns`: Het aantal patrnen (cases) in de dataset.

`fmatrix *orig_patterns`: `fmatrix` voor de originele patronen.

`fmatrix *stan_patterns`: `fmatrix` gestandaardiseerde patronen. Bij gestandaardiseerde patronen heeft iedere numerieke variabele via een transformatie een gemiddelde van 0 en een standaarddeviatie van 1 gekregen.

`fmatrix *covmat`: Covariantiematrix van de numerieke variabelen.

`nominal_var **labels`: Tweedimensionaal array voor nominale variabelen.

`float *up`: `float` array dat hoogste waarde in dataset bevat voor iedere numerieke variabele.

`float *stan_up`: `float` array dat hoogste waarde in gestandaardiseerde dataset bevat voor iedere numerieke variabele.

`float *low`: `float` array dat laagste waarde in dataset bevat voor iedere numerieke variabele.

`float *stan_low`: `float` array dat laagste waarde in gestandaardiseerde dataset bevat voor iedere numerieke variabele.

`float *avg`: `float` array dat gemiddelde waarde in dataset bevat voor iedere numerieke variabele.

`float *stdv`: `float` array dat standaard-deviatie bevat voor iedere numerieke variabele.

`int var_types[255]`: `int` array dat van iedere variabele in dataset aangeeft wat het type is, waarbij geldt dat numerieke variabelen met een 0 gecodeerd zijn, nominale variabelen met een 1.

`nominal_var *var_names`: Array van `nominal_var`'s dat de namen van de variabelen bevat.

Member functions

`dataset()`: Default constructor. Dient niet te worden aangeroepen.

`dataset(TNVDataset *deze, AnsiString, bool[])`: Dit moet Jeroen nog even invullen.

`char* input_file_name()`: Functie die de naam van het bestand (file) oplevert die gebruikt is om de dataset te creëren.

`void compute_covariance_matrix()`: Berekent de covariantiematrix van de numerieke variabelen in de dataset.

`~dataset()`: Destructor function.

Hierin is het datatype `nominal_var` een zelf gedefinieerde klasse die er als volgt uit ziet:

```
class nominal_var
{
  // overloading of the input operator
  friend ifstream& operator>> (ifstream&, nominal_var&);

  public:
    char label[50];
};
```

3.2 De `sanger` *class voor Sanger neuraal netwerk*

De `sanger` class is geïmplementeerd in de bestanden `sanger.h` en `sanger.cpp`. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables De allocatie van geheugen voor matrices, vectors etc. vindt plaats in de constructor function van de sanger class.

`fmatrix *mappings`: `fmatrix` die de afbeeldingen van de gestandaardiseerde patronen uit de dataset waarop het Sanger neuraal netwerk getraind is bevat.

`fmatrix *norm_mappings_2d`: `fmatrix` die de eerste twee dimensies van de afbeeldingen van de gestandaardiseerde patronen uit de dataset waarop het Sanger neuraal netwerk getraind is bevat. Deze dimensies zijn zodanig geschaald dat ze zich tussen 0 en 1 bevinden. Deze 2-koloms matrix wordt op het scherm getekend.

`fvector *activations:` `fvector` die de activaties van de uitvoerunits van het Sanger neuraal netwerk bevat bij het aanbieden van één patroon aan de invoerzijde.

`fvector *scree_vector:` `fvector` die de waarden bevat die in het scree-diagram op het scherm weergegeven worden.

`float lrate:` Leersnelheid parameter voor Sanger neuraal netwerk.

Member functions

`sanger():` Default constructor. Dient niet te worden aangeroepen.

`sanger(dataset*):` Constructor met een pointer naar dataset object als argument. Het sanger object dat gecreëerd wordt slaat deze dataset op in een private member variable.

`void compute_activations(fvector& fv):` Functie die de uitvoer van het Sanger neuraal netwerk berekent wanneer `fvector fv` aan de invoerzijde aangeboden wordt.

`void init():` Initialiseert het Sanger neuraal netwerk (o.a. random initialisatie gewichten).

`void train(int nof_cycles):` Traint het Sanger neuraal netwerk `nof_cycles` iteraties. In één iteratie worden alle patronen uit de dataset die bij het aanmaken van het Sanger neuraal netwerk gebruikt is in willekeurige volgorde een maal aangeboden. Na ieder patroon worden de gewichten aangepast.

`void compute_vector_image(fvector& iv, fvector& ov):` Berekent voor invoervector `iv` het beeld waarop deze wordt afgebeeld. Dit beeld wordt opgeslagen in `fvector ov`.

`void norm_vector_2d(fvector& ov):` Normaliseert de eerste twee elementen van `fvector ov` op dezelfde manier als `norm_mappings_2d` is genormaliseerd.

`~sanger():` Destructor function.

3.3 De `ica` *class voor ICA neuraal netwerk*

De `ica` class is geïmplementeerd in de bestanden `ica_net.h` en `ica_net.cpp`. Om een zekere uniformiteit te berwerkstelligen zijn zoveel mogelijk namen gelijk gekozen aan de namen die in de `sanger` klasse gebruikt zijn. De public interface van de `ica_net` klasse bevat de volgende member variables en member functions.

Member variables De allocatie van geheugen voor de hieronder genoemde matrices en vectoren vindt plaats in de constructor function. De identifiers zijn analoog gekozen aan de gebruikte namen voor matrices, vectoren etc. in het artikel waarin de methode geïntroduceerd is. Dit artikel is

J. Karhunen, E. Oja, L. Wang, R. Vigário, and J. Joutsensalo. A class of neural networks for independent component analysis. *IEEE Transactions on Neural Networks*, 8(3):486–504, May 1997.

Uit dit artikel is het *bigradient* algoritme (eqn. 18) gebruikt.

`fmatrix *v_mappings:` `fmatrix` die data na 'PCA whitening' bevat.

`fmatrix *y_mappings:` `fmatrix` die data na independent component analysis bevat.

`fmatrix *V:` `fmatrix` die transformatiematrix bevat die nodig is om 'PCA whitening' tot stand te brengen. Deze matrix brengt tevens een dimensionaliteitsreductie tot stand.

`fmatrix *V_COMPLETE:` `fmatrix` die transformatiematrix bevat die nodig is om 'PCA whitening' tot stand te brengen. Deze matrix brengt geen dimensionaliteitsreductie tot stand en wordt daarom 'de complete versie van V' genoemd.

`fmatrix *W`: `fmatrix` die transformatiematrix bevat die nodig is om independent component transformatie tot stand te brengen. Deze matrix wordt ook 'separation matrix' genoemd.

`fmatrix *WT`: `fmatrix` die getransponeerde van `W` bevat.

`fmatrix *Q`: `fmatrix` die transformatiematrix bevat die nodig is om data terug te transformeren van independent components naar de originele data.

`fmatrix *I`: `fmatrix` die identiteitsmatrix bevat.

`fmatrix *D`: `fmatrix` die diagonaalmatrix bevat met op de diagonaal de eigenwaarden van de covariantiematrix - nodig voor PCA whitening.

`fmatrix *SQRTD`: `fmatrix` die diagonaalmatrix bevat met op de diagonaal de wortels uit de eigenwaarden van de covariantiematrix - nodig voor PCA whitening.

`fmatrix *SQRTDINV`: `fmatrix` die inversematrix van `*SQRTD` bevat - nodig voor PCA whitening.

`fmatrix *E`: `fmatrix` die als kolommen de eigenvectoren van de covariantiematrix heeft - nodig voor PCA whitening.

`fmatrix *ET`: `fmatrix` die getransponeerde van `E` bevat.

`fmatrix *tmpmat0`: `fmatrix` die gebruikt wordt als tijdelijke opslag in berekeningen.

`fmatrix *tmpmat1`: `fmatrix` die gebruikt wordt als tijdelijke opslag in berekeningen.

`fmatrix *tmpmat2`: `fmatrix` die gebruikt wordt als tijdelijke opslag in berekeningen.

`fmatrix *x`: `fmatrix` die één patroon uit de dataset kan bevatten.

`fmatrix *v`: `fmatrix` die de afbeelding van één patroon uit de gestandaardiseerde dataset kan bevatten na PCA whitening.

`fmatrix *y`: `fmatrix` die de afbeelding van één patroon uit de gestandaardiseerde dataset kan bevatten na independent component analysis.

`fmatrix *yt`: `fmatrix` die getransponeerde van `fmatrix *y` bevat.

`fmatrix *gyt`: `fmatrix` die identiek is aan `fmatrix *yt`, maar met functie g toegepast op ieder element.

`fmatrix *norm_mappings_2d`: `fmatrix` die de eerste twee dimensies van de afbeeldingen van de gestandaardiseerde patronen uit de dataset waarop het ICA neuraal netwerk getraind is bevat. Deze dimensies zijn zodanig geschaald dat ze zich tussen 0 en 1 bevinden. Deze 2-koloms matrix wordt op het scherm getekend.

`fvector *scree_vector`: `fvector` die de waarden bevat die in het scree-diagram op het scherm weergegeven worden.

`int L`: Aantal invoerknopen, uitvoerknopen van het ICA neuraal netwerk. Dit aantal is gelijk aan het aantal numerieke variabelen in de dataset die gebruikt is om het ICA netwerk aan te maken.

`int M`: Aantal knopen in de twee verborgen lagen van het ICA netwerk. Dit aantal is gelijk aan het aantal 'independent components' dat men wil extraheren. In het geval van NeuralVision is dit altijd twee.

Member functions

`ica_net()`: Default constructor voor `ica_net` class. Dient niet te worden aangeroepen.

`ica_net(dataset*,int)`: Constructor met een pointer naar een dataset object als argument. Het ica_net object dat gecreëerd wordt slaat deze dataset op in een private member variable.

`void init()`: Initialiseert het ICA neuraal netwerk (o.a. random initialisatie gewichten).

`void estimate_V()`: Functie die de schatting van de 'PCA whitening matrix' `V` uitvoert.

`void estimate_W(int)`: Functie die de schatting van de 'separation matrix' `W` uitvoert.

`void estimate_Q()`: Functie die de schatting van de 'mixing matrix' `Q` uitvoert.

`void compute_v_mappings()`: Functie die de afbeeldingen van de gestandaardiseerde patronen uit de dataset volgens de 'PCA whitening matrix' `V` berekent en opslaat in `fmatrix *v_mappings`.

`void compute_y_mappings()`: Functie die de afbeeldingen van patronen in `fmatrix *v_mappings` volgens de 'separation matrix' berekent en opslaat in `fmatrix *y_mappings`. Dit zijn de statistisch onafhankelijke signalen.

`void compute_vector_image(fvector& iv, fvector& ov)`: Berekent voor invoervector `iv` het beeld waarop deze wordt afgebeeld door het ICA netwerk. Dit beeld wordt opgeslagen in `fvector ov`.

`void norm_vector_2d(fvector&)`: Normaliseert de eerste twee elementen van `fvector ov` op dezelfde manier als `norm_mappings_2d` is genormaliseerd.

`~ica_net()`: Destructor function.

3.4 De `fscnn` *class voor fscnn neuraal netwerk*

De `fscnn` class is geïmplementeerd in de bestanden `fscnn.h` en `fscnn.cpp`. De public interface van deze klasse bevat de volgende member variables en member functions.

Member variables Geheugen voor matrices, vectoren etc. wordt in de constructor van deze klasse gealloceerd.

`fmatrix *weights`: `fmatrix` die de gewichtenvectoren van de units uit het fscnn netwerk bevat.

`fmatrix *variances`: `fmatrix` die de varianties bevat binnen de groepen die door de fscnn units worden gerepresenteerd.

`int nof_units`: `int` die het aantal knopen in het netwerk bevat.

`float error`: Gemiddelde afstand tot het bijbehorende cluster-centrum (fscnn gewichten vector) van de patronen in de gestandaardiseerde dataset.

Member functions

`fscnn()`: Default constructor. Dient niet te worden gebruikt.

`fscnn(dataset*, int n_units)`: Constructor met een dataset object als argument. Het fscnn object dat gecreëerd wordt slaat deze dataset op in een private member variable. `n_units` geeft het aantal units aan.

`void init()`: Initialiseert het fscnn neuraal netwerk (o.a. random initialisatie gewichten).

`void train(int nof_cycles)`: Traint het fscnn neuraal netwerk `nof_cycles` cycles.

`~fscnn()`: Destructor function.

4. Gebruik van de NV2 specifieke classes

4.1 De `dataset` *class voor inlezen dataset en initiële bewerking daarvan*

Om een dataset te creëren dient de constructor function van de dataset class te worden aangeroepen met de bestandsnaam van een databestand. Een voorbeeld hiervan is

```
dataset *d;

/*
 * create a dataset object
 */
d = new dataset("iris.ndf");
```

Vervolgens kan men met behulp van een call naar `compute_covariance_matrix()` de covariantiematrix van het numerieke deel van de ingelezen dataset laten berekenen:

```
/*
 * compute the sample covariance matrix
 */
d->compute_covariance_matrix();
```

In NV2 wordt de covariantiematrix gebruikt in de `ica_net` class.

4.2 De `sanger` *class voor Sanger neuraal netwerk*

De `sanger` class gebruikt men als volgt. Eerst maakt men een `sanger` object aan met de call

```
sanger *sn;

/*
 * create sanger object
 */
sn = new sanger(d);
```

In deze call wordt een pointer naar een `dataset` object meegegeven als argument. Vervolgens wordt het `sanger` object geïnitialiseerd met behulp van de call

```
/*
 * init sanger object
 */
sn->init();
```

Het aangemaakte netwerk kan nu getraind worden met de call

```
/*
 * train sanger object
 */
sn->train(100);
```

Desgewenst kan de `lrate` member variable aangepast worden. Deze heeft standaard een waarde van 0.1 en wordt automatisch verlaagd. Na het trainen zijn de `mappings`, `norm_mappings_2d`, `scree_vector` ingevuld met de correcte waarden.

4.3 De `ica` *class voor ICA neuraal netwerk*

Wederom geldt hier dat eerst een `ica_net` object aangemaakt dient te worden. Dit gebeurt met de function call

```
ica_net *ica;

/*
 * create ica_net object
 */
ica = new ica_net(d,2);
```

In deze call worden een pointer naar een `dataset` object en een getal dat het aantal gewenste onafhankelijke componenten weergeeft meegegeven als argument. Vervolgens wordt het `ica_net` object geïnitialiseerd met behulp van de call

```
/*
 * init ica object
 */
ica->init();
```

Hierna kan de berekening van de eigenlijke ICA projectie beginnen. Deze bestaat uit een een aantal stappen

1. Het schatten van de 'PCA whitening' matrix `V`. Dit gebeurt met behulp van de function call

   ```
   // estimate the pca matrix V
   ica->estimate_V();
   ```

   In deze fase wordt de `scree_vector` berekend.

2. Het berekenen van de 'whitened data' `v_mappings`. Dit gebeurt met behulp van de function call

   ```
   // compute whitened data
   ica->compute_v_mappings();
   ```

3. Het schatten van de 'separation' matrix `W`. Dit is de trainingsfase van net NN algoritme en wordt uitgevoerd door de function call

   ```
   // estimate the separation matrix W (NN training)
   ica->estimate_W(30);
   ```

   Het argument (in dit geval 30) geeft het gewenste aantal iteraties aan. Tijdens deze fase worden automatisch de onafhankelijke data `y_mappings` geschat.

4. Het schatten van de 'mixing' matrix `Q`. Dit gebeurt met behulp van de function call

   ```
   // estimate the mixing matrix Q
   ica->estimate_Q();
   ```

   Voor data-projectie doeleinden (dus ook in NV2) is deze stap niet noodzakelijk.

4.4 De `fscnn` *class voor fscnn neuraal netwerk*
Wederom geldt hier dat eerst een `fscnn` object aangemaakt dient te worden. Dit gebeurt door middel van de function call

```
// create fscnn object
fn = new fscnn(d);
```

In deze call wordt een pointer naar een `dataset` object meegegeven als argument. Vervolgens wordt het `fscnn` object geïnitialiseerd met behulp van de call

```
// init fscnn object
fn->init();
```

Het netwerk kan nu getraind worden met de function call

```
// train fscnn object
fn->train(100);
```

Na het trainen bevatten `weights` en `variances` de gewenste cluster-centra en varianties binnen de clusters.

5. Toelichting bij de genomen ontwerp-beslissingen

De ondersteunende classes zijn ontworpen en geïmplementeerd om het programmeren van de NV2 specifieke classes eenvoudiger te maken.

Voor de NV2 specifieke classes zijn grotendeels onafhankelijk van elkaar geïmplementeerd, met uitzondering van de `dataset` class, die in alle neurale netwerk classes gebruikt wordt. Er is geen gebruik gamaakt van inheritance. Aangezien de drie geïmplementeerde neurale netwerken erg van elkaar verschillen is dit een logische beslissing.