Chapter 1

General Neural Network Framework

Our first effort will be to implement a general framework to implement and execute Neural Network (NN) experiments in *FastDownward*.

1.1 Training-Evaluation Implementation

Training Scheme:

```
define/load network N
while some condition C_T:
  sample training data (FastDownward) or load some data
  train network
  (Optionally) store network
store network and/or provide object pointer
```

Listing 1.1: Training Phase

Evaluation Scheme:

```
load network N/get pointer of N
while searching:
  # do search
  give state to N (convert format as necessary)
  obtain results (convert format as necessary)
  # continue search
```

Listing 1.2: Evaluation Phase

For the training scheme i can imagine the following implementations:
Patrick

I think we should plan on the implementation **both**, but start with the *Python* version. The *C++* version cannot be done without major changes in *FastDownward*. Although, in the last *FastDownward* meeting the idea

Implementation	Advantages	Disadvantages
In *FastDownward* (pure C++)		-good *C++* NN library? -changing goal not supported now (on todo list) -needs something encapsulating the training searches which is currently not thought of now -*FastDownward* initializes state space at start up, thus cannot be changed later [on todo list]
In *Python* (except *FastDownward* for sampling)	-many powerful libraries	-slow data flow between *Python* and *FastDownward* (network serialization, training data) [is intended to change by implementing a Python API, but before this can be implemented, the todos in *FastDownward* disadvantages have to be solved]
both	-maximal flexibility for user	-implement both version and solve all problems of both versions

Table 1.1: Advantages/Disadvantages of approaches

for a Python API had positive feedback and for this we would also do the here required reworks, I have no clue how long this rework will take. The *Python* version can already be implemented. As *Python* is today majorily used for deep learning, we have a huge arsenal of libraries available and some should support also *C++* (see section 1.2 "Neural Network Libraries"). If it is implemented as new module to the driver scripts of *FastDownward*, then the user does not feel a difference to a normal *FastDownward* command (except for new arguments). A drawback is that the information flow between *Python* and *C++* will be relatively slow compared to pure *C++*. This might improve if the *Python* API is implemented, but still then there will be an overhead for passing the objects around.

Thus, I think we should focus on the *Python* version, because we can already do this, but when implementing it, we should already think on what the *C++* would need. For example, the command line arguments should be the same such that the user will not notice a difference and the sampling in *FastDownward* should support being called from the outside (by *Python*) and from within (by the *C++* version).

1.2 Neural Network Libraries

Name	Languages	Comments
Tensorflow	*Python*, *C++*	-training only in *Python* possible

Table 1.2: Neuronal Network Libraries

1.3 Neural Network Implementation

Patrick

The chosen training scheme restricts our freedom of library choices, but still multiple libraries might be possible. I suggest structuring the code in *FastDownward* in such a way that it can be extended for any library. A possible structure can be seen in Figure 1.1.

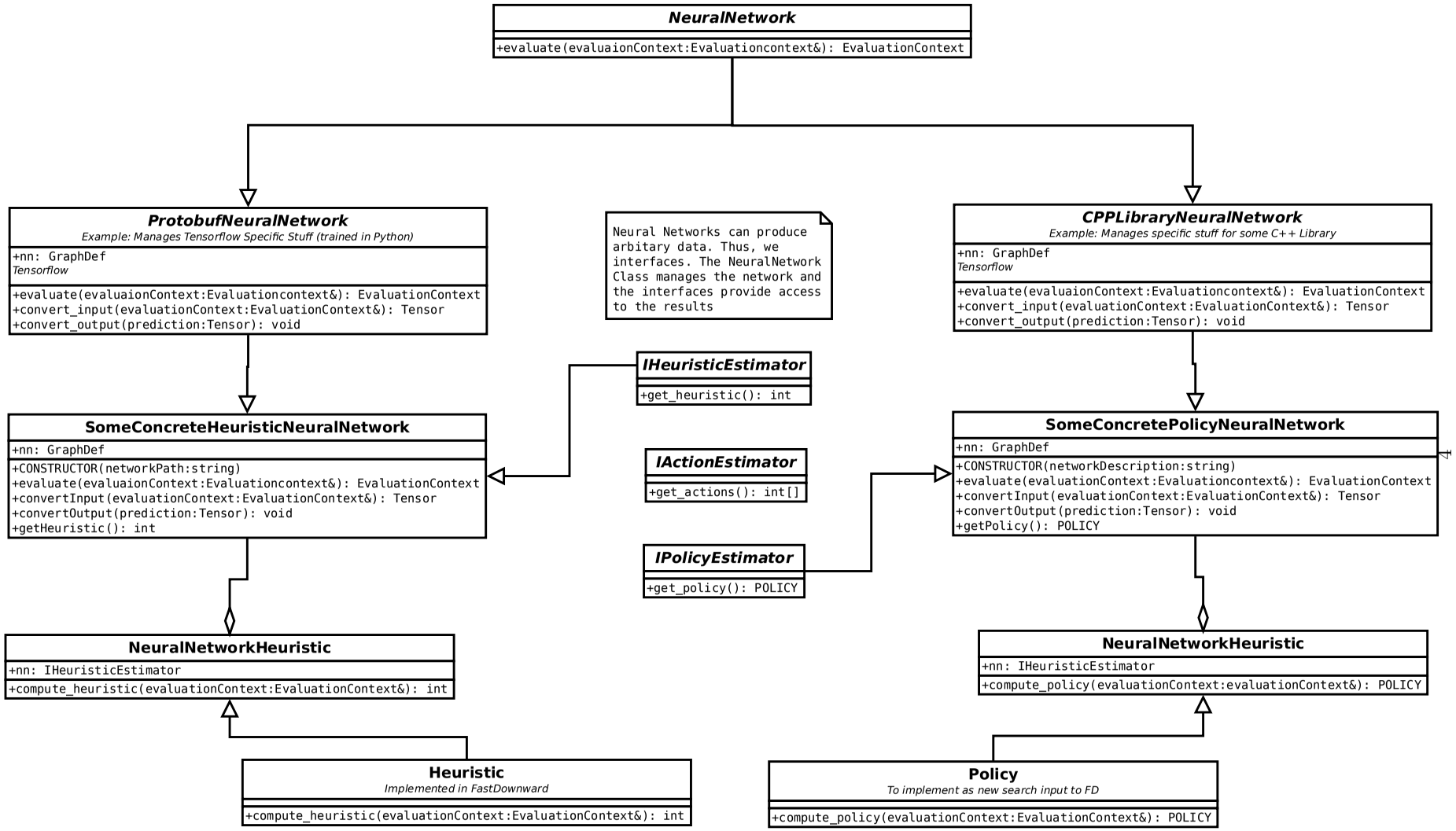
An abstract NN base class provides the *evaluate* method which takes as input the data provided by *FastDownward* for a state evaluation. A library should subclass this and implement its own management parts. The example shows a class for Protobuf (which is the format used by tensorflow to store graphs). A concrete NN usable by the user would subclass the base class of its library. NNs can output very different things, thus, the networks *evaluate* does not return anything, but instead it shall set member variables. A concrete network implements then some interfaces for the output it produces (it might produce multiple different outputs) and his interface methods shall provide access to the output of the NN associated to the interface (e.g. a heuristic value). New interfaces can be added for new kinds of output of NNs.

1.4 Policies

A new thing we might want to implement in *FastDownward* would be policies. A policy tells the search which actions to choose in a state together with a rating for the action.

Are Preferred Operators policies with equal rating among all chosen actions?

Most times NNs which output an action to chose actually output actions with a rating, therefore, could be directly used as policy.


NeuralNetwork
+evaluate(evaluaionContext:Evaluationcontext&): EvaluationContext
ProtobufNeuralNetwork
Example: Manages Tensorflow Specific Stuff (trained in Python)
+nn: GraphDef
Tensorflow
+evaluate(evaluaionContext:Evaluationcontext&): EvaluationContext
+convert_input(evaluationContext:EvaluationContext&): Tensor
+convert_output(prediction:Tensor): void
Neural Networks can produce arbitary data. Thus, we interfaces. The NeuralNetwork Class manages the network and the interfaces provide access to the results
CPPLibraryNeuralNetwork
Example: Manages specific stuff for some C++ Library
+nn: GraphDef
Tensorflow
+evaluate(evaluaionContext:Evaluationcontext&): EvaluationContext
+convert_input(evaluationContext:EvaluationContext&): Tensor
+convert_output(prediction:Tensor): void
IHeuristicEstimator
+get_heuristic(): int
SomeConcreteHeuristicNeuralNetwork
+nn: GraphDef
+CONSTRUCTOR(networkPath:string)
+evaluate(evaluaionContext:Evaluationcontext&): EvaluationContext
+convertInput(evaluationContext:EvaluationContext&): Tensor
+convertOutput(prediction:Tensor): void
+getHeuristic(): int
IActionEstimator
+get_actions(): int[]
IPolicyEstimator
+get_policy(): POLICY
SomeConcretePolicyNeuralNetwork
+nn: GraphDef
+CONSTRUCTOR(networkDescription:string)
+evaluate(evaluationContext:Evaluationcontext&): EvaluationContext
+convertInput(evaluationContext:EvaluationContext&): Tensor
+convertOutput(prediction:Tensor): void
+getPolicy(): POLICY
NeuralNetworkHeuristic
+nn: IHeuristicEstimator
+compute_heuristic(evaluationContext:EvaluationContext&): int
NeuralNetworkHeuristic
+nn: IHeuristicEstimator
+compute_policy(evaluationContext:evaluationContext&): POLICY
Heuristic
Implemented in FastDownward
+compute_heuristic(evaluationContext:EvaluationContext&): int
Policy
To implement as new search input to FD
+compute_policy(evaluationContext:EvaluationContext&): POLICY