# Praktikum Adaptive Systeme WS 14 - Einführung

## 1. Zum Inhalt des Praktikums

In diesem Praktikum werden wir uns mit verschiedenen Arten von Künstlichen Neuronalen Netzen beschäftigen und ihre Verwendung zur Informationsverarbeitung, zur Analyse von Daten und zum Lösen komplexerer Probleme untersuchen. Im Vordergrund stehen wird dabei die Fähigkeit dieser Strukturen, eine Lösung zu einem vorgegebenen Problem selbständig zu erlernen. In diesem Zusammenhang werden wir uns dann mit verschiedenen Lernalgorithmen beschäftigen und uns sowohl Netze zur Lösung allgemeiner Probleme, als auch spezielle auf die Problemstellung zugeschnittene Netze ansehen.
Zu Beginn des Praktikums wird eine kurze Einführung stehen (dieses Dokument und das erste Aufgabenblatt), welches einige Grundlagen zu biologischen und formalen (künstlichen) Neuronen liefert und einige Aufgaben zum „Warmwerden“ mit formalen Neuronen und der Verwendung der Entwicklungsumgebung (Eclipse) enthält.
Am Ende des Praktikums wird es dann einen kleinen Wettbewerb geben, bei dem Sie über einen Zeitraum von etwa drei Wochen selbständig die im Verlauf des Praktikums erworbenen Fähigkeiten zur Analyse unbekannter Daten anwenden können.

## 2. Die Entwicklungsumgebung

### 2.1. Einrichtung

Als Entwicklungsumgebung verwenden wir **Eclipse**, in welcher die gestellten Aufgaben in der Programmiersprache **Python** umgesetzt werden sollen. Downloaden Sie dafür Python für Ihr Betriebssystem (empfehlenswert ist die Version **2.7.x**, da einige existierende 3rd-party Software mit Version 3.3 noch nicht kompatibel ist). Außerdem benötigen Sie das **PyDev** Plug-in für Eclipse. Auf den Rechnern im Praktikumsraum ist dies bereits installiert, wenn Sie auf Ihrem Rechner arbeiten wollen, sollten Sie der Anleitung unter http://pydev.org/manual_101_install.html folgen.

Konfiguration des Interpreters für PyDev in Eclipse

**Window -> Preferences -> PyDev -> Interpreter - Python -> New** (siehe Abbildung 1)
Hier können auch alle andere Einstellungen angepasst werden, beispielsweise vom Editor (General -> Editors), Konsole (Run/Debug -> Console) etc.

Weitere nützliche Pakete müssen installiert werden:

- **NumPy** und **SciPy** – erleichtert die Arbeit mit mathematischen Strukturen wie Vektoren, Matrizen etc. und Operationen (http://www.numpy.org/und http://scipy.org/).
- **MatPlotLib –** Bibliothek für die graphische Darstellung von Funktionen, Punktmengen, Geraden und ähnlichem… (http://matplotlib.org/).
  Tutorial unter http://matplotlib.org/users/pyplot_tutorial.html

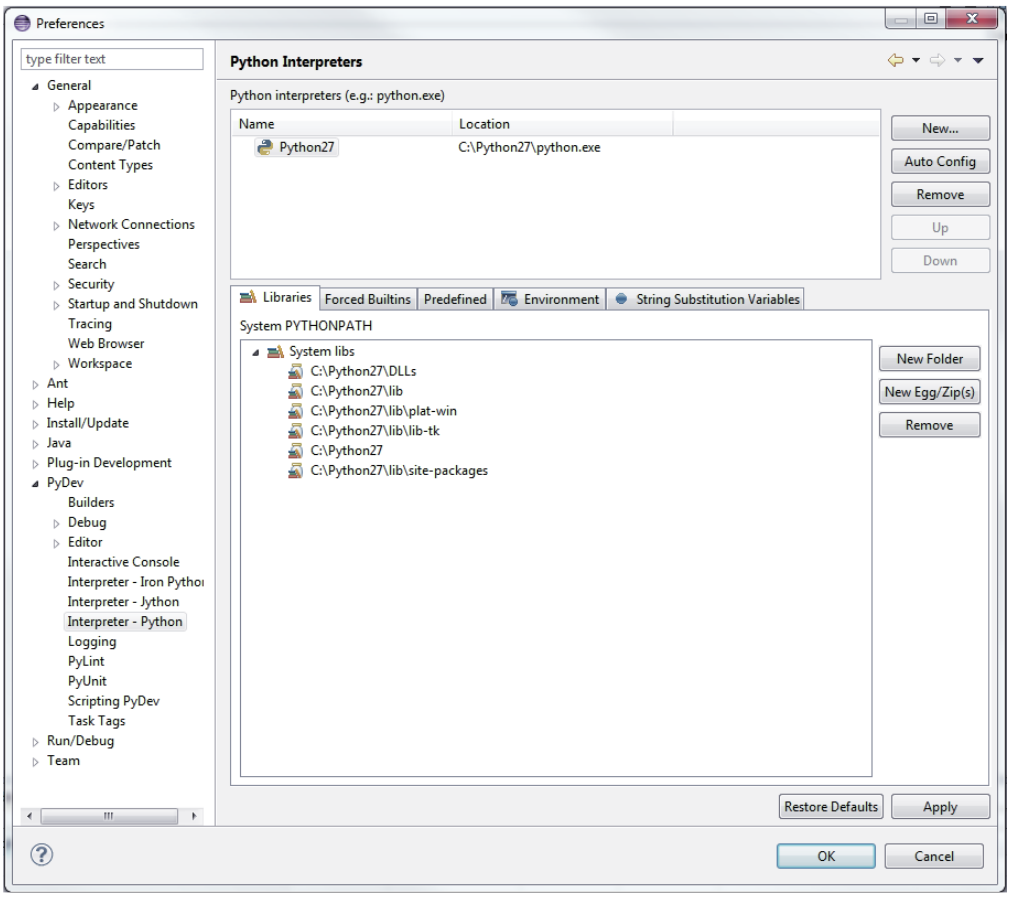

Abbildung 1: PyDev - Konfiguration

**Nützliche Methoden in Python und Numpy:**

- `numpy.loadtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)[source]`

Load data from a text file. Each row in the text file must have the same number of values.

-> **Beispiel**: data = numpy.loadtxt("digits.dt")
for j in range(data.shape[1]): # iteriere durch die Spalten
adaline.training(data[:,j]) # alle Zeilen der Spalte j

- `numpy.reshape(a, newshape, order='C')`

Gives a new shape to an array without changing its data.

-> **Beispiel**: >>> a = np.array([[1,2,3], [4,5,6]])
>>> np.reshape(a, (3,-1)) # -1 ist ein unspezifizierter Wert, ergibt 2 bei 3 Zeilen
array( [[1, 2],
[3, 4],
[5, 6]] )

- `random.sample(population, k)`

Return a *k* length list of unique elements chosen from the population sequence. Used for random sampling without replacement.

- `numpy.dot(a, b, out=None)`

Dot product of two arrays.

-> Notwendig für Produkt zwischen Matrix-Vektor, Vektor-Vektor etc.

- `numpy.transpose(A)` oder `A.T` -> Transponierte der Matrix A

- `numpy.outer(a, b)`

Compute the outer product of two vectors.

- `numpy.array_equiv(a1, a2)[source]`

Returns True if input arrays are shape consistent and all elements equal.

- `numpy.zeros(shape=(x,y))`
  -> Praktisch für die Initialisierung einer Matrix der Form (x,y), um später die berechnete Matrixelemente einzusetzen.

-> In manchen Situationen - numpy.ones(shape=(x,y))

- `defaultdict(list)`

-> Dictionary-Struktur mit Möglichkeit, die Listen als dict-Value abzuspeichern, Beispiele auf http://docs.python.org/release/2.5.2/lib/defaultdict-examples.html.

### 2.2. Projekt-Erstellung

Ein neues Projekt unter Eclipse erstellen Sie mit:
**File -> New -> PyDev-Project**
Nachdem Sie hier einen Namen für Ihr Projekt vergeben haben, geben Sie als Interpreter Python27 an und klicken Sie auf **Finish**. Das angegebene Projekt wird dann unter Eclipse als Ordner im Fenster „PyDev Package Explorer“ angelegt. Mittels
**Rechte Maustaste -> New ->PyDev Package**
auf den Projektordner kann dort ein Klassenpacket angelegt werden, in welches Sie mit
**Rechte Maustaste -> New -> PyDev Module**
neue Klassen für Ihr Programm hinzufügen können. Beachten Sie, dass in Python keine Definition der main()-Funktion als Haupteintrittspunkt ihres Programmes angelegt wird.
Man startet das Modul von der Konsole aus oder man verwendet dessen Attribut **`__name__`** im IF-Block wie folgt: `if __name__ == '__main__'`.

Hierzu finden Sie nützliche Informationen unter:
http://stackoverflow.com/questions/419163/what-does-if-name-main-do.

## 3. Was gehört zu einer erfolgreichen Bearbeitung der Aufgaben?

Zu Beginn wird jede Woche ein Aufgabenblatt ins Netz gestellt, für dessen Bearbeitung Sie bis zur Ausgabe des nächsten Blattes, also genau eine Woche, Zeit haben. Spätere Aufgabenblätter (in der Mitte des Praktikums) werden etwas umfangreicher sein oder mehr Zeit zur Bearbeitung benötigen, so dass hier ein längerer Zeitraum von zwei bzw. drei Wochen vorgesehen ist.

Die Lösung der gestellten Aufgaben besteht aus einem lauffähigen Programm und der zugehörigen Dokumentation.
Die Aufgaben gelten als erfolgreich gelöst, wenn die zugehörigen Programme lauffähig sind, die in der Aufgabenstellung geforderte Funktionalität erfüllen und die Anforderungen an die Dokumentation erfüllt sind, sowie die Abgabe rechtzeitig bis spätestens zum auf dem Aufgabenzettel angegebenen Termin erfolgt ist.
Die Abgabe der fertigen Programme (bestehend aus den kommentierten Quellcodedateien) und der Dokumentation (vorzugsweise als PDF) erfolgt dabei als ZIP-File.
Nach welchen Kriterien generell die Benotung der Abgaben erfolgt, können Sie dabei unter „Bewertung“ bei den einzelnen Aufgabenblättern erkennen.
Wird eine Aufgabe nicht fristgerecht gelöst (essentielle Dinge sind nicht erstellt oder funktionieren nicht, die Vorführung/Erklärung der Arbeit wird vom Tutor nicht mit „ausreichend“ bewertet), so erhält das Teammitglied/das Team die Möglichkeit, die Korrektur zusätzlich zu den neuen Aufgaben in der darauffolgenden Woche abzugeben. Ist eine solche Nachbesserung während des gesamten Praktikums zwei Mal erfolgt, so wird beim dritten Mal das Praktikum für „nicht bestanden“ erklärt. Dies bedeutet auch, dass ein zweimaliges Fehlen beim Praktikum noch toleriert wird, jedoch kein drittes Mal. Erledigt ein Teammitglied in diesem Zusammenhang seine Arbeit in der Gruppe nicht ausreichend, so wird dies nur ihm angerechnet, nicht aber der Gesamtgruppe.
Eine Abmeldung vom Praktikum kann innerhalb der ersten zwei Wochen erfolgen, ohne dass dies als „nicht bestanden“ gewertet wird.

Generell gelten noch einige allgemeine Bedingungen, die an jede Abgabe gestellt werden:

a) **Nachschlagen unbekannter Begriffe**
   Sobald Sie einen Begriff sehen, der Ihnen unbekannt vorkommt, wird von Ihnen erwartet, dass Sie diesen nachschlagen und sich über das Thema informieren. Selbstredend sollte dies nach mindestens zwei Semestern Studium der Informatik selbstverständlich sein.

b) **Informieren über Themen**
   Genauso, wie wir von Ihnen erwarten, dass Sie unbekannte Begriffe nachschlagen, erwarten wir von Ihnen auch, dass Sie sich das Wissen, welches in den einzelnen Aufgabenblättern benötigt wird, mit Hilfe des Materials, das wir Ihnen zur Verfügung stellen, oder auch den entsprechenden Büchern, selbst aneignen. Empfehlungen sowie Links finden Sie auf der Homepage des Praktikums. Bei spezielleren Problemen können Sie sich natürlich auch gerne an den Tutor wenden.

c) **Kommentare**
   Längere Codestellen, die nicht in Methoden ausgelagert werden können, sollten durch die Verwendung von Kommentaren in sinnvolle Einheiten unterteilt werden. Die Kommentare sollen dabei erklären, was eine einzelne dieser Einheiten tut, bzw. was genau ihre Funktion ist. Beachten Sie, dass Personen allein an diesen Kommentaren verstehen sollten, was Ihr Quellcode eigentlich genau macht. Als Sprache der Kommentare sollte Englisch verwendet werden, da dies die in der Informatik am meisten verwendete Sprache ist. Natürlich ist auch Deutsch okay. Sie sollten dies allerdings einheitlich machen und innerhalb derselben Klasse immer dieselbe Sprache verwenden.

d) **Dokumentation**

Die Dokumentation ist ein wichtiger Teil des Praktikums. Eine detaillierte Beschreibung der Anforderungen an die Dokumentation finden Sie im Aufgabenverzeichnis unter „Hinweise zur Dokumentation“, in dem Ihnen auch die zu bearbeitenden Aufgaben zur Verfügung gestellt werden.

**e) Quellenangaben**
Falls Sie externe Quellen verwenden, so geben Sie diese Quellen an. Bedenken Sie: bei aller Nützlichkeit ist Wikipedia leider keine Quelle, die für wissenschaftliche Arbeit benutzt werden kann.

**f) Kein „Copy & Paste“**
Stimmt die Lösung einer Gruppe in charakteristischen Merkmalen mit der einer anderen Person bzw. Gruppe überein, so gilt sie als kopiert. Kopierte Lösungen werden natürlich nicht als Aufgabenlösung akzeptiert, unabhängig davon, wer Urheber und wer Plagiator ist.

**g) Präsentationsgespräch**
Beim Präsentationsgespräch wird die Lösung der Aufgabe von den einzelnen Teammitgliedern präsentiert. Jedes Mitglied sollte über alle Teile, auch die, die es nicht angefertigt hat, Bescheid wissen und sie präsentieren können.

Bei der Erstellung von Programmen empfiehlt es sich außerdem, unbedingt auf eine sinnvolle Einteilung der Klassen und Methoden und eine klar auskommentierte Programmstruktur zu achten, da dies die Übersichtlichkeit, Testbarkeit und Wartbarkeit von Programmen enorm erhöht und zur Vermeidung von Fehlern beiträgt (und darüber hinaus zu einem guten Programmierstil gehört). Gute Programme sind soweit wie möglich in weitestgehend allgemein verwendbare und unabhängige Programmkomponenten zerlegt (modularer Aufbau).