# NNW-Übung 4

## Abgabe

Bitte laden Sie den/die Quellcode(s) als *einzelne* Python (.py) Datei(en) oder Jupyter-Notebook(s) (.ipynb) einschließlich aller benötigten Datendateien und weiteren Python-Quellcode-Dateien (wie z.B. `nnwplot`) ab (optional gerne auch ein aus dem Jupyter-Notebook generiertes PDF), also nicht gezippt. Der abgegebene Quellcode sollte sich ohne Änderungen ausführen lassen. Die Aufgabennummern bitte im Quellcode als Kommentare (oder Markdown) angeben und Fragen ebenfalls direkt im Quellcode beantworten.

## 1 Einstieg in Keras: Klasse `Sequential`

a) Importieren Sie Keras mit `from tensorflow import keras`.

b) Implementieren Sie zunächst ein einschichtiges Netz zur Klassifikation aller drei (!) Iris-Arten, jetzt aber in Keras und zunächst mit Hilfe der Klasse `keras.models.Sequential` und ihrer Methode `add(layer)`.

Benutzen Sie die letzten beiden Merkmale (bei Zählung ab 0 also die Merkmale 2 und 3).

**Achtung**: Anders als bisher müssen bei Keras die Trainingsdaten in den Zeilen (statt Spalten) stehen!

Sie brauchen dafür[1]:

- Ein Objekt von der Klasse `keras.models.Sequential`; dieses Objekt wird normalerweise `model` genannt.
- Die Methode `add(layer)` von `Sequential`, um Layers hinzuzufügen. Die hinzugefügten Layers werden nacheinander ausgeführt, mit den Ausgaben der vorigen Layer als Input der nächsten.
- `keras.layers.Dense(units,activation=None,input_shape=None,name=None)` um fully-connected („dicht“ verbundene) Schichten zu erstellen:
  - `units`: Anzahl der Units der Layer.
  - `activation`: Aktivierungsfunktion als String mit dem Namen der Funktion oder Keras-Objekt. Für die Ausgabeschicht `'softmax'` verwenden.
  - `input_shape`: wird nur in der ersten Schicht gebraucht und muss dort als Tupel die Dimension *eines* Eingabedatums (Featurevektors) enthalten.
  - `name`: Optional: String mit Namen der Schicht um darauf zugreifen zu können und zur Anzeige in TensorBoard.
- Die Methode `compile(optimizer, loss=None, metrics=None)` von `Sequential` zum Konfigurieren des Lernvorgangs:
  - `optimizer`: Optimierer als String mit dem Namen der Funktion oder Keras-Objekt, z.B. `'adam'`
  - `loss`: Fehlerfunktion als String mit dem Namen der Funktion oder Keras-Objekt, z.B. `'mean_squared_error'`.
  - `metrics`: Zusätzlich auszuwertende Funktionen zur Bewertung des Fehlers durch den Benutzer als Liste von Strings mit dem Namen der Funktionen oder Keras-Objekte, z.B. „`['accuracy']`“.

[1] Die Parameterlisten sind im Folgenden immer auf das Wesentliche gekürzt – für alle Parameter siehe die Keras Dokumentation

- Die Methode `fit(x=None, y=None, epochs=1)` von `Sequential` zum Durchführen des Trainings:
  - `x`: Numpy Array mit den Trainingsdaten. **Achtung**: Anders als bisher müssen bei Keras die Trainingsdaten in den Zeilen (statt Spalten) stehen!
  - `y`: Numpy Array mit den Zieldaten. Für Klassifikation: Zielwerte one-hot kodieren. Hierfür gibt es die Keras-Funktion `keras.utils.to_categorical`. **Achtung**: Auch hier: die Zieldaten müssen in den Zeilen (statt Spalten) stehen!
  - `epochs`: Anzahl Epochen, die das Training durchgeführt werden soll.
- Die Methode `predict(x)` von `Sequential` zum Ausführen des Netzes auf den Eingabedaten `x`.

c) Plotten Sie das Ergebnis wie gehabt mit der Funktion `nnwplot.plotTwoFeatures`. Hier auch die Daten in Zeilen (statt Spalten) übergeben.

d) Lassen Sie sich die Anzahl der Gewichte ausgeben. Die Methode dazu finden Sie zum Beispiel im Foliensatz NNW_05b_KerasCodeAusschnitte.

e) Erweitern Sie das Netz um eine Hidden-Layer mit fünf Neuronen und `tanh` Aktivierungsfunktion. Trainieren Sie das Netz und lassen Sie sich das Ergebnis wieder mit der Funktion `plotTwoFeatures` plotten. Lassen Sie sich erneut die Anzahl der Gewichte ausgeben und vollziehen Sie das Ergebnis nach.

## 2 Einstieg in Keras: Klasse `Model`

a) Implementieren Sie wieder ein Netz mit *zwei* Schichten analog zur vorigen (Teil-)Aufgabe zur Klassifikation der drei Iris-Arten, jetzt aber mit Hilfe der Klasse `keras.models.Model`.

*Wichtig*: Vorteil dieser Vorgehensweise ist, dass sie komplexere Netzwerkstrukturen (z.B. mit „Verzweigungen“) erlaubt.

Sie brauchen dafür:

- Ein Objekt von der Klasse `keras.Input(shape)`, wobei `shape` ein Tupel mit der Dimension *eines* Eingabedatums (Featurevektors) ist.
- Layer-Objekte, also hier wie gehabt `keras.layers.Dense`, diesmal diese aber als Objekt angeben und *direkt danach* in Klammern die Daten übergeben, die von der Schicht verarbeitet werden sollen, z.B. `b = keras.layers.Dense(32)(a)`, wenn `a` die Daten sind, also entweder das im vorigen Schritt angelegte `Input`-Objekt oder die Ausgaben der voherigen Schicht. Die Ausgaben dieser Schicht ist im Beispiel `b`.
- Ein Objekt von der Klasse `keras.Model(inputs, outputs)`, das normalerweise `model` genannt wird, mit
  - `inputs`: das oben angelegte Input-Objekt.
  - `outputs`: das Ausgabe-Objekt der letzten Schicht.
- Die Methoden `compile`, `fit` und `predict` analog zu voriger Aufgabe auf dem Objekt von `Model` aufrufen.

b) Plotten Sie wieder das Ergebnis mit der Funktion `nnwplot.plotTwoFeatures`.

c) Lassen Sie ein Plot der Netzwerkarchitektur erstellen. Wie das geht, finden Sie zum Beispiel im Foliensatz NNW_05b_KerasCodeAusschnitte – das gilt auch für die folgenden Teilaufgaben!

d) Plotten Sie nach Ende des Trainings den Verlauf der Werte der Fehlerfunktion und der Accuracy im Laufe des Trainings.

e) Fügen Sie einen Callback hinzu, der nach je 10 Epochen die Zwischengebnisse mit `plotTwoFeatures` plottet.