NNW-Übung 5 (optional)

Backpropagation klassisch: Training eines zweischichtigen Netzes

In dieser Aufgabe soll ein zweischichtiges Netz mit Backpropagation trainiert werden – klassich mit selbst programmierter Ableitung, also ohne automatisches Ableiten, ohne ein spezielles Framework und ohne einen Computational Graph.

Die benötigten Formeln finden Sie im Foliensatz „04a Backpropagation für Neuronale Netze“ – Sie werden sehen, mehr als Addieren und Multiplizieren (und manchmal noch Ableitung der Aktivierungsfunktion) ist für Backpropagation nicht notwendig.

- Vollziehen Sie den Code der Datei `NNW_P5_A1_unvollstaendig.py` nach, der ein einfaches zweischichtiges Netzes implementiert.

  Wichtig: Beachten Sie insbesondere die verwendete Aktivierungsfunktion in der Hidden-Layer (wichtig für die nächste Teilaufgabe).

- Ergänzen Sie die Zeilen, in denen Text statt Code steht.

- Berechnen Sie die Zwischenergebnisse (wie `delta1`, `delta2`)

  1.) per Matrixmultiplikationen ...

  2.) **oder**, falls Sie die vielen Indizes doch lieber einfach abtippen möchten, mit Hilfe der Funktion *Einsteinschen Summenkonvention* unter Verwendung von deren Implementierung in Numpy als `np.einsum`: Über doppelte Indizes wird summiert. Beispiele:

     - Matrix `A` mal Vektor `v` ist damit `np.einsum('ij,j', A, v)` – das Komma markiert, was sich auf die erste und was auf die zweite übergebene Variable bezieht, der nicht doppelte Index `i` ist der Index der Komponenten des Ergebnisvektors.
     - Matrix `A` mal Matrix `B` ist `np.einsum('ij,jk', A, B)` – die nicht doppelten Indizes `i,k` sind die Indizes der `i`-ten Zeile/`j`-ten Spalte der Ergebnismatrix.
     - Skalarprodukt von Vektor `a` und Vektor `b` ist `np.einsum('i,i',a,b)`
     - Transponieren der Matrix `M` `np.einsum('ij->ji', M)`
     - Weitere Beispiele auf der Numpy-Seite zu `np.einsum` und – etwas übersichtlicher – zu der entsprechenden Funktion von PyTorch torch.einsum.