KI und neuronale Netze

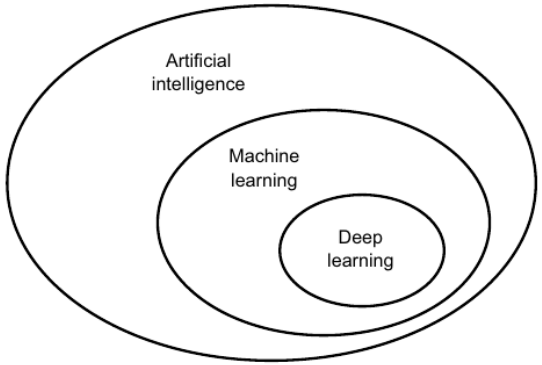
Olaf Schnürer

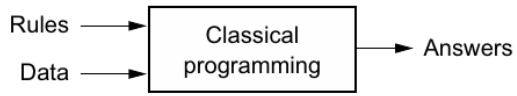
Seminar zur Fachdidaktik Mathematik, PH Thurgau, Juni 2023

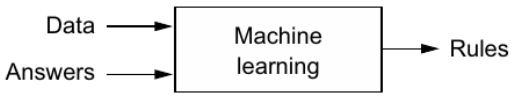
Künstliche Intelligenz

Mögliche Definition

Der Versuch, von Menschen erledigte Aufgaben automatisiert zu lösen.







Geschichte

- ab 1950er Jahren vor allem *symbolische KI*: regelbasierte Programme, ELIZA, erste Schachprogramme, Expertensysteme
- ab 1980er Jahren Aufschwung von *machine learning*: Computer lernt Regeln aus Trainingsdaten

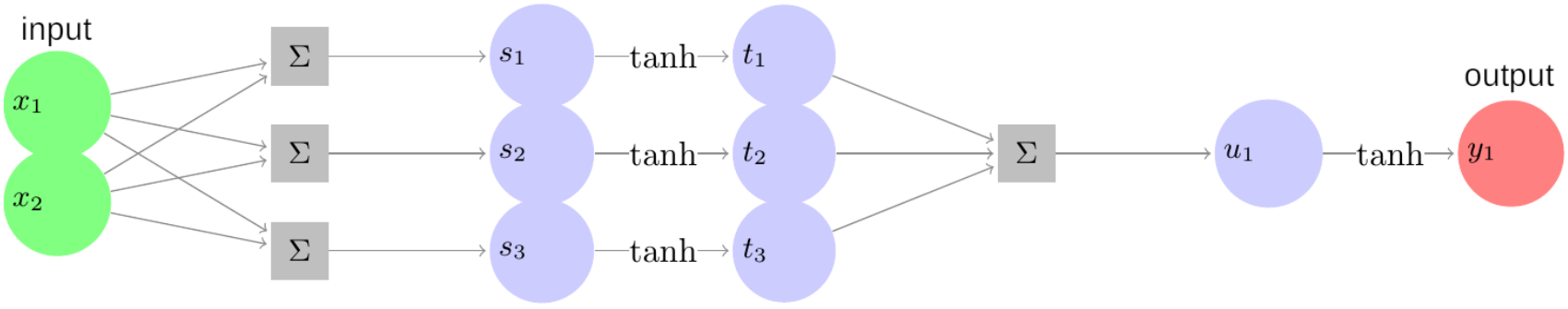
  (vgl. frühkindliches Sprachenlernen vs. Grammatik-Regeln)
- ab ca. 2010 *deep learning* „technisch realisierbar“: grosse neuronale Netze mit vielen (daher *deep*) Schichten

Grundlegende Theorie alt! mind. seit 1940er, 1950er Jahren

Erfolge

Auf nahezu menschlichem Niveau: Bildklassifizierung, Handschriftenerkennung, Übersetzung, selbstfahrende Autos; bester menschlicher Go-Spieler besiegt (AlphaGo, 2016); Proteinstrukturvorhersage (AlphaFold); Beantworten natürlichsprachiger Fragen (ChatGPT); Text- und Bildproduktion (etwa im Stil von van Gogh).

Beispiel eines neuronalen Netzes

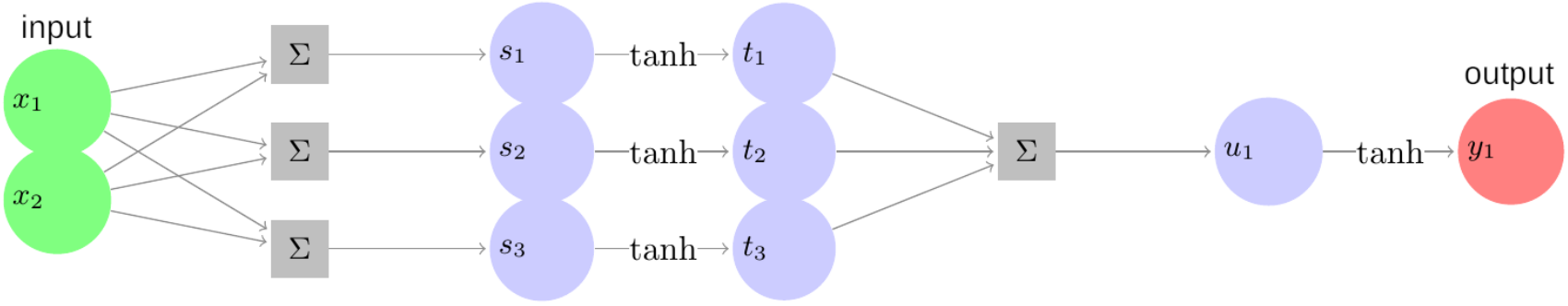


- Input-Vektor (oder besser Input-Vektorraum)
- Output-Vektor
- 3 versteckte Vektoren (hidden vectors)
- 4 Schichten (layers); hier: Schicht = Übergang(sfunktion) von einem Vektor zum nächsten (Der Begriff Schicht wird teilweise auch für die Vektoren gebraucht.)

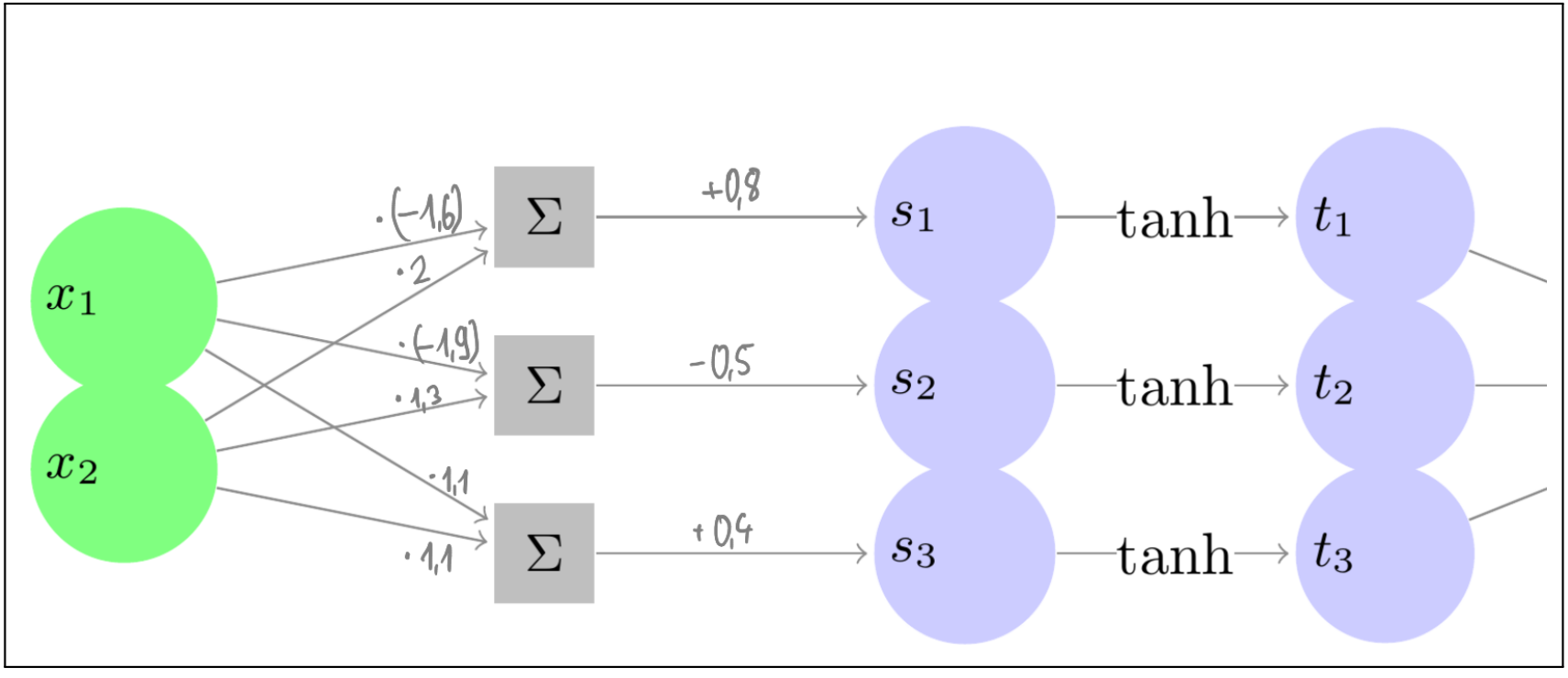
Idee

Die im Input enthaltene Information wird schichtweise verändert. So erhält man in den hidden vectors neue Darstellungen (von Teilaspekten) der Information. Am Ende ist hoffentlich die gesuchte Information als Output extrahiert.

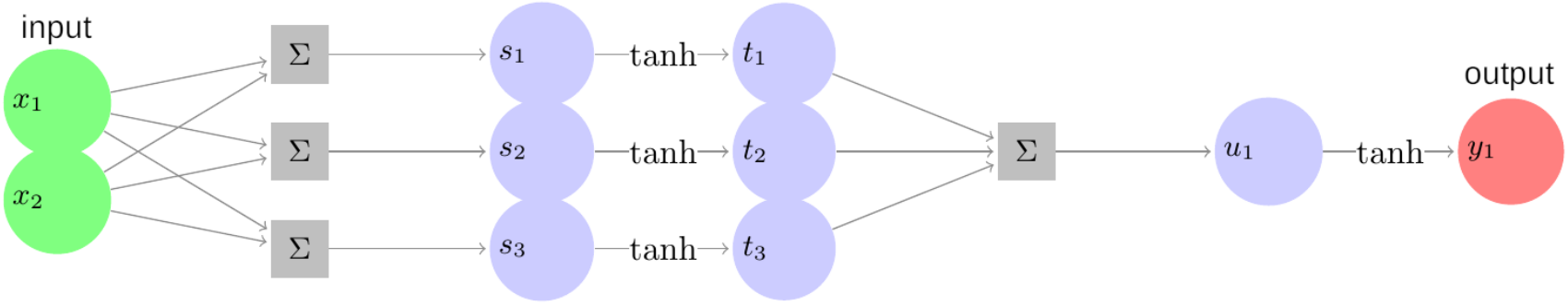
Beispiel eines neuronalen Netzes



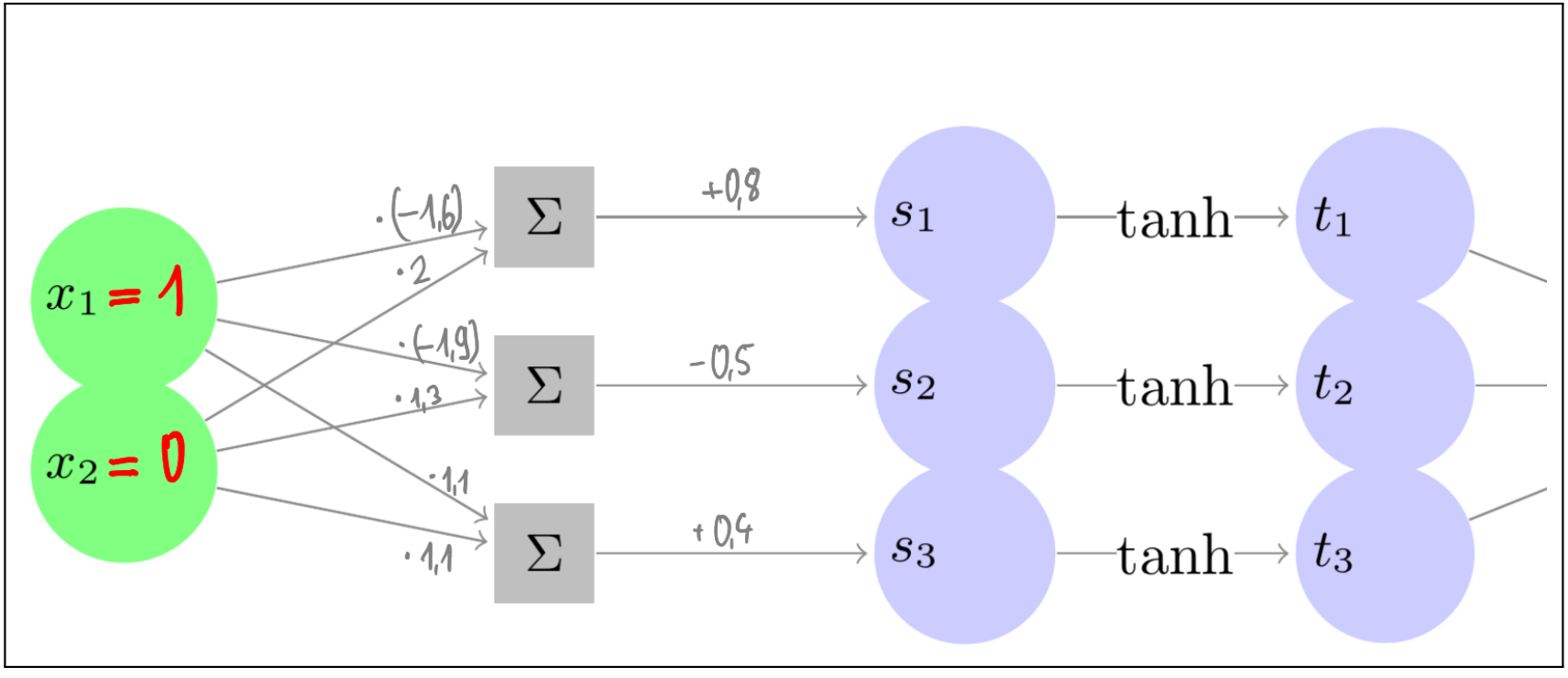
Ausschnitt



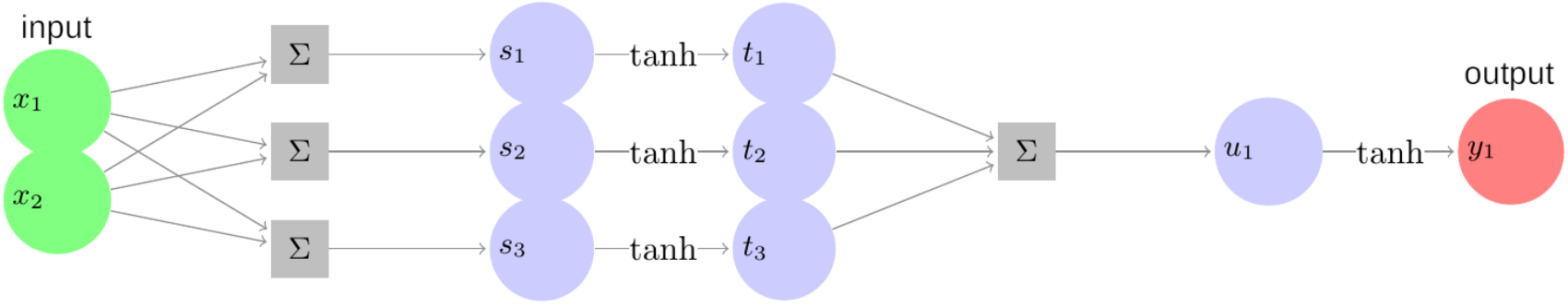
Beispiel eines neuronalen Netzes



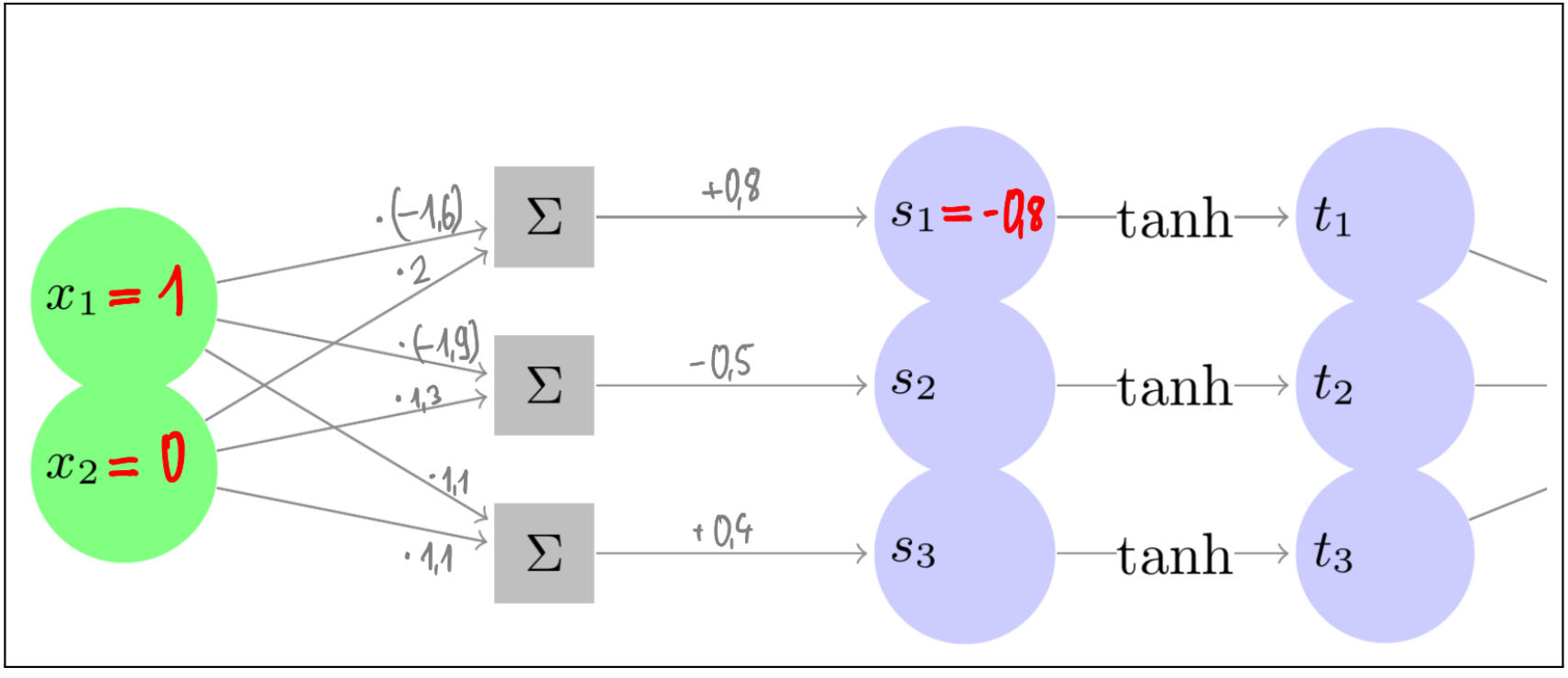
Ausschnitt



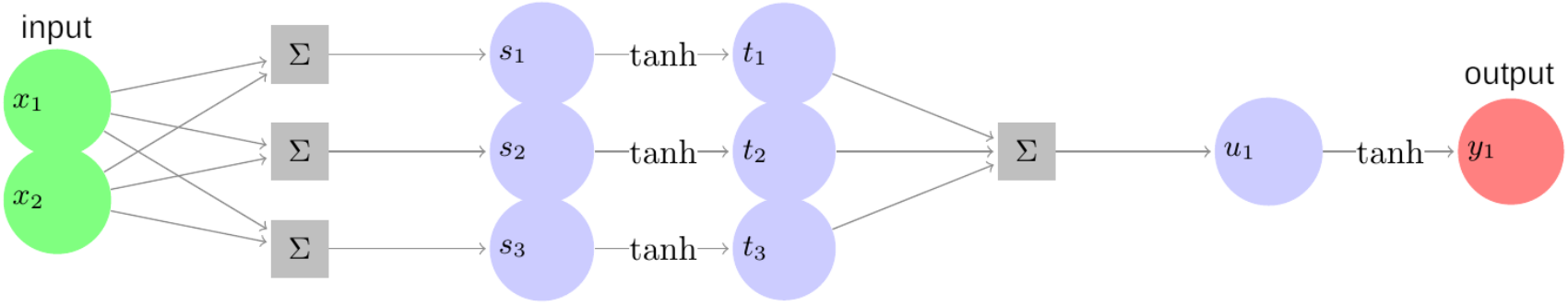
Beispiel eines neuronalen Netzes



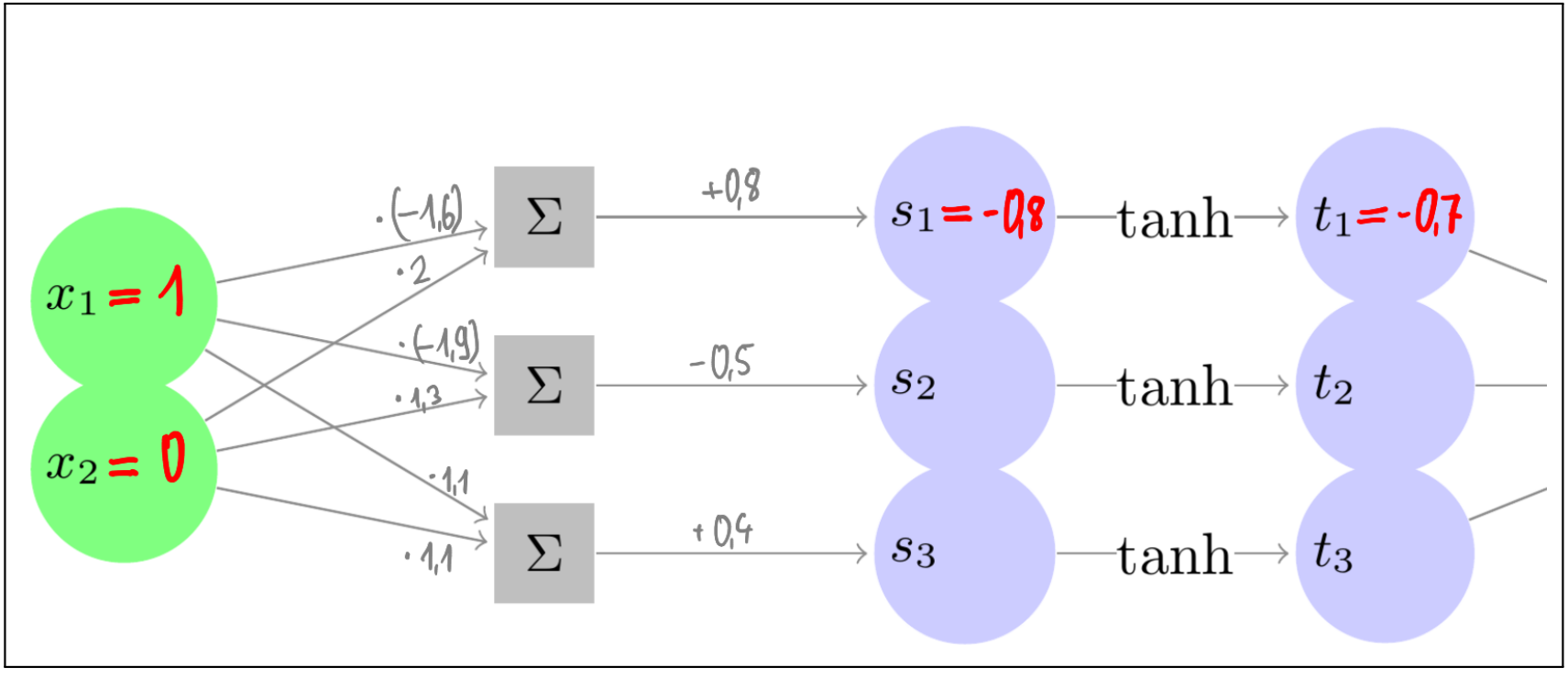
Ausschnitt



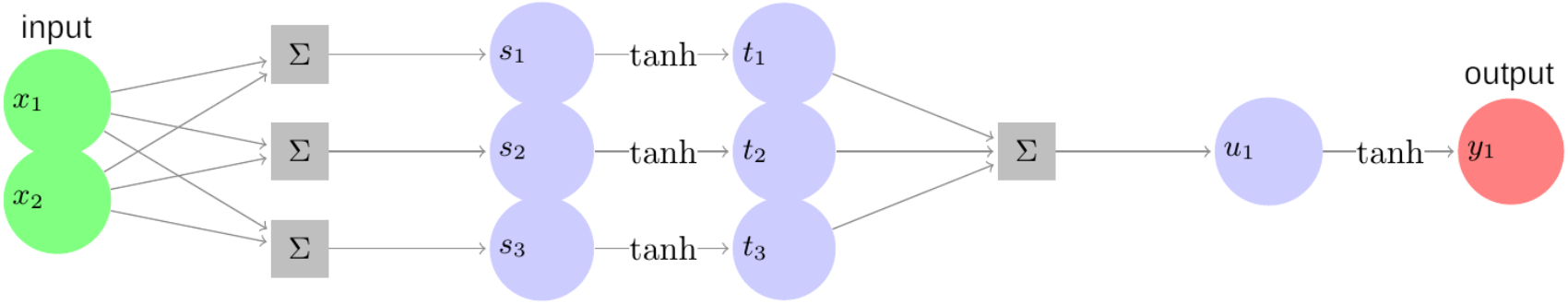
Beispiel eines neuronalen Netzes



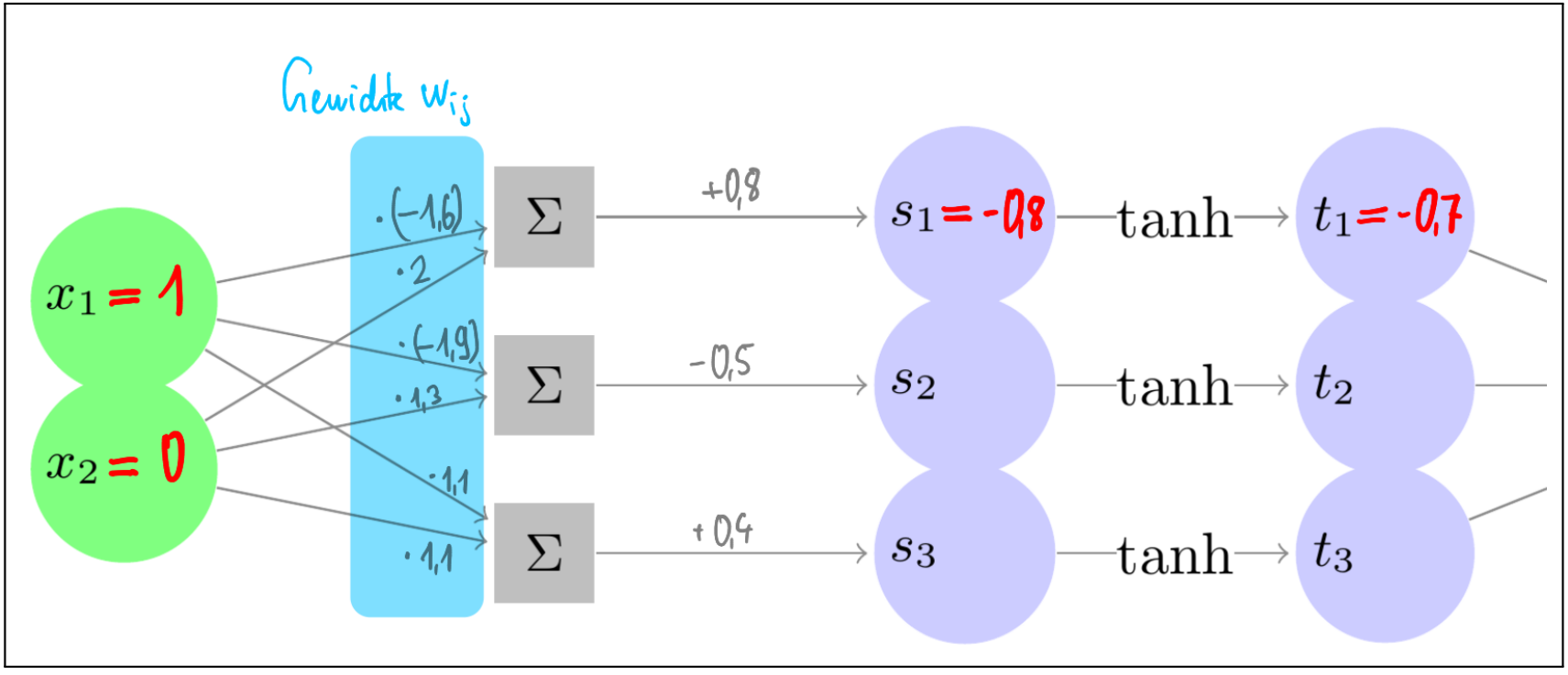
Ausschnitt



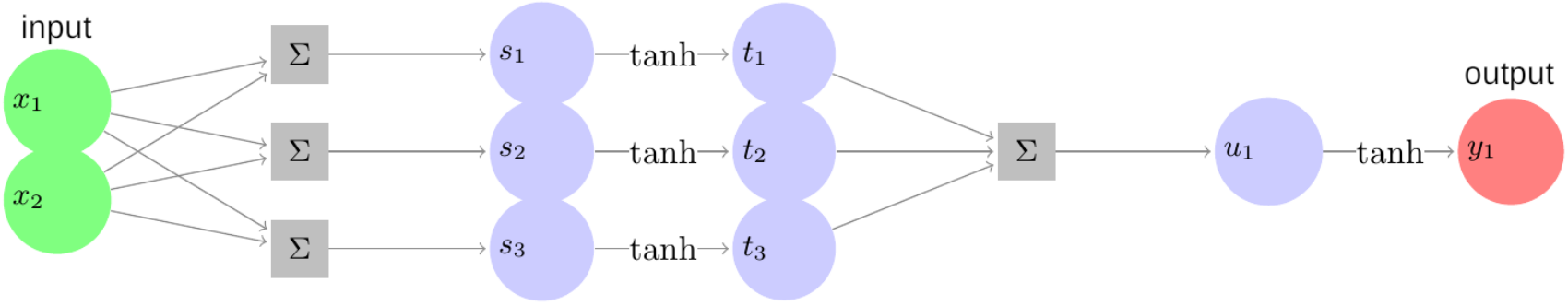
Beispiel eines neuronalen Netzes



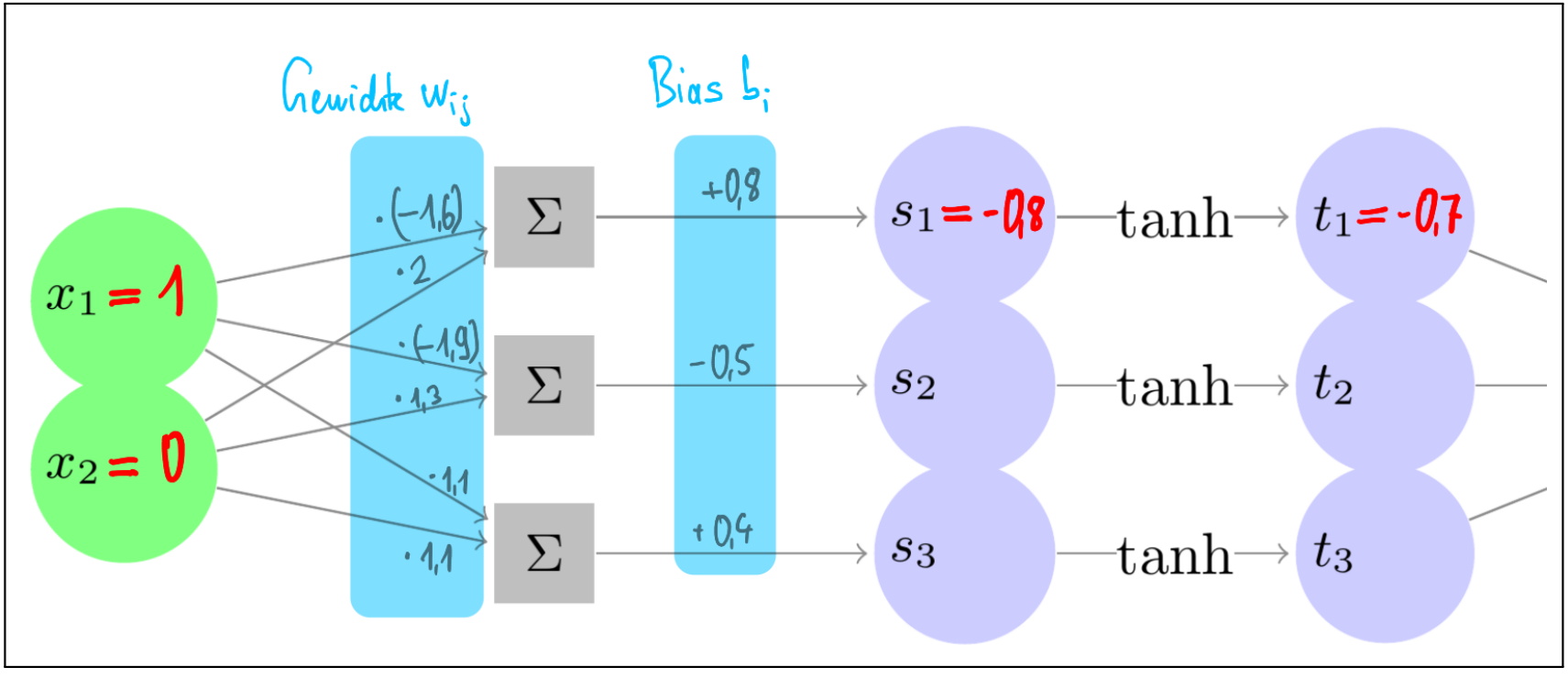
Ausschnitt



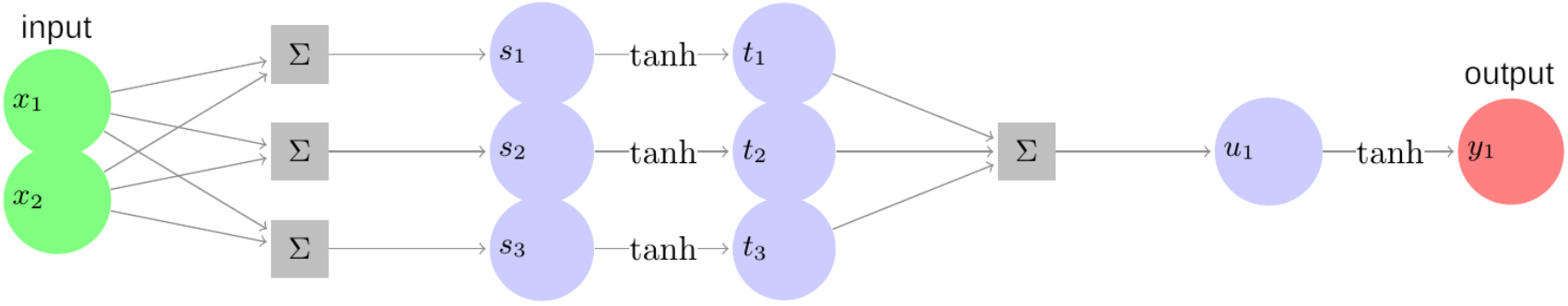
Beispiel eines neuronalen Netzes



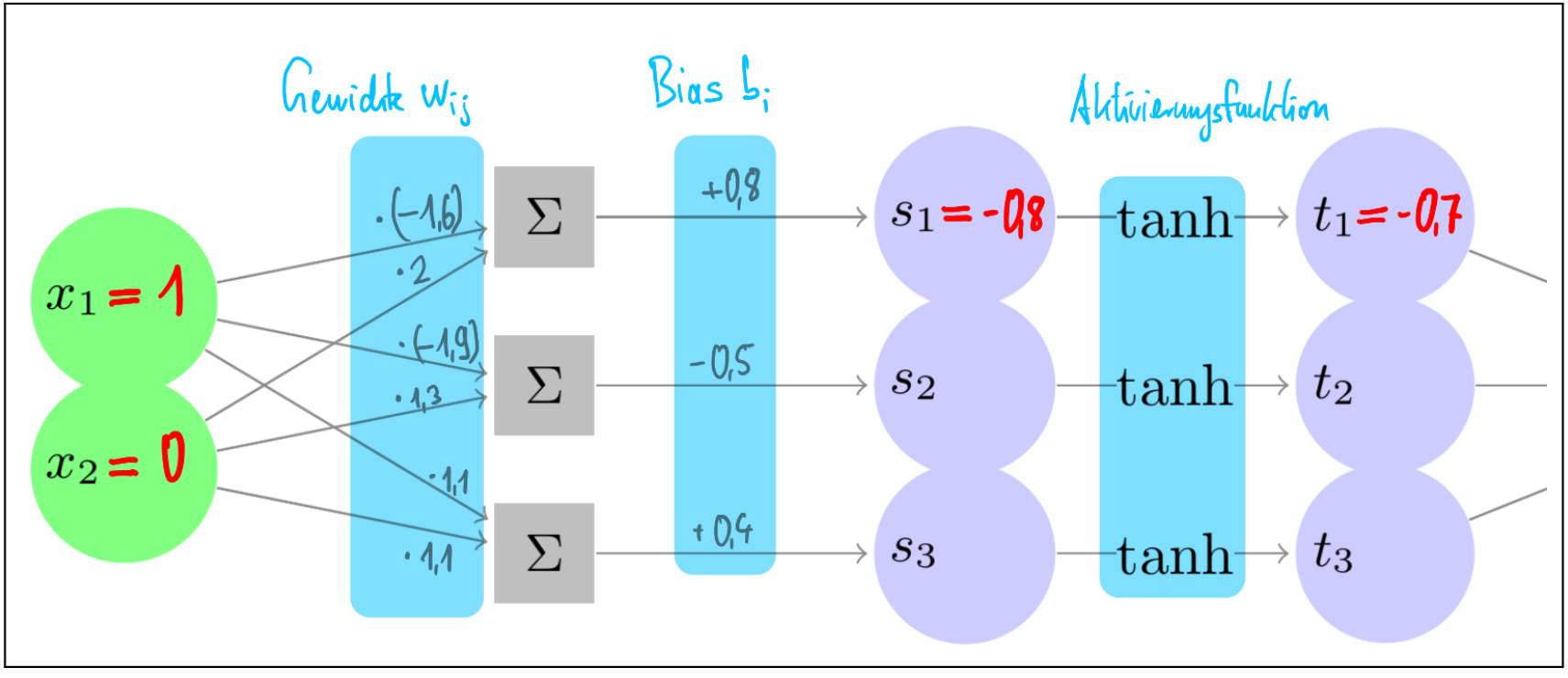
Ausschnitt



Beispiel eines neuronalen Netzes

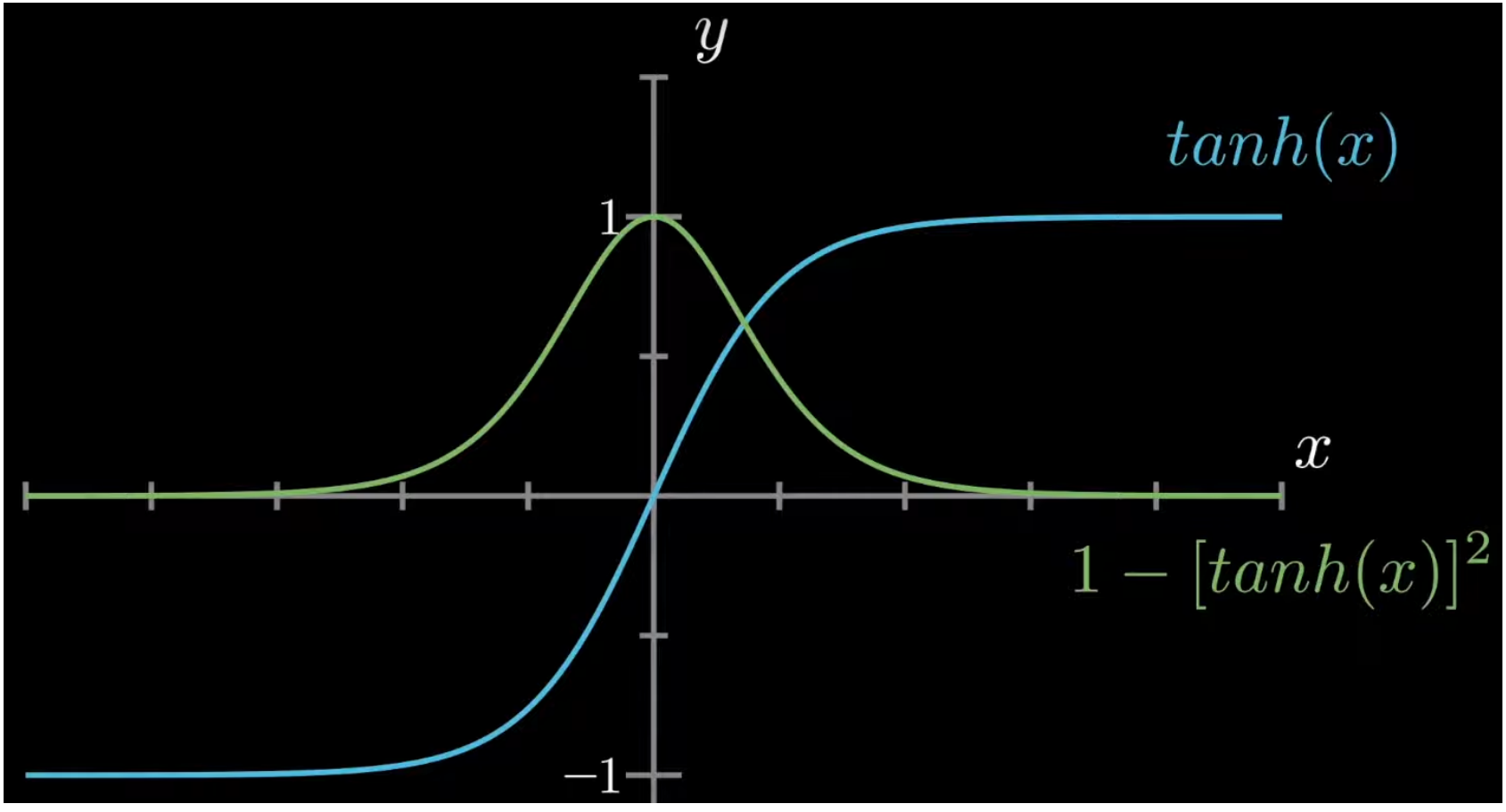


Ausschnitt

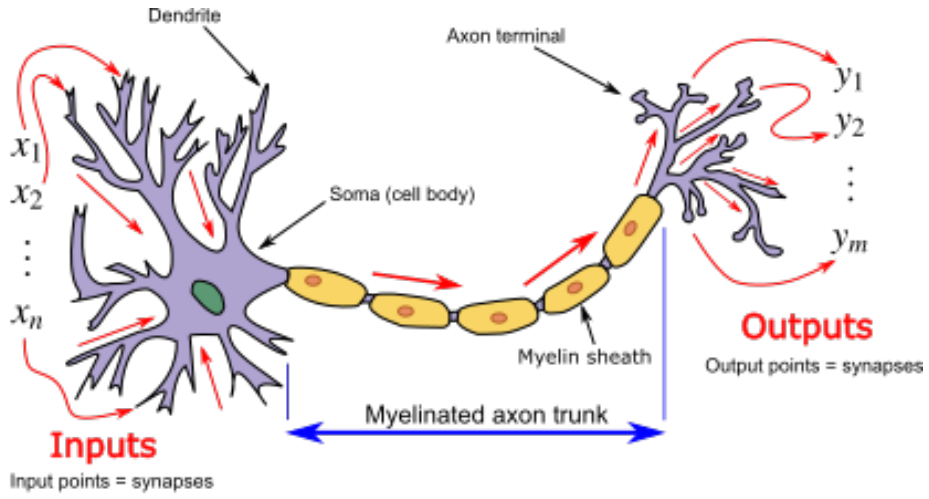


Exkurs: Aktivierungsfunktion (samt Ableitung)

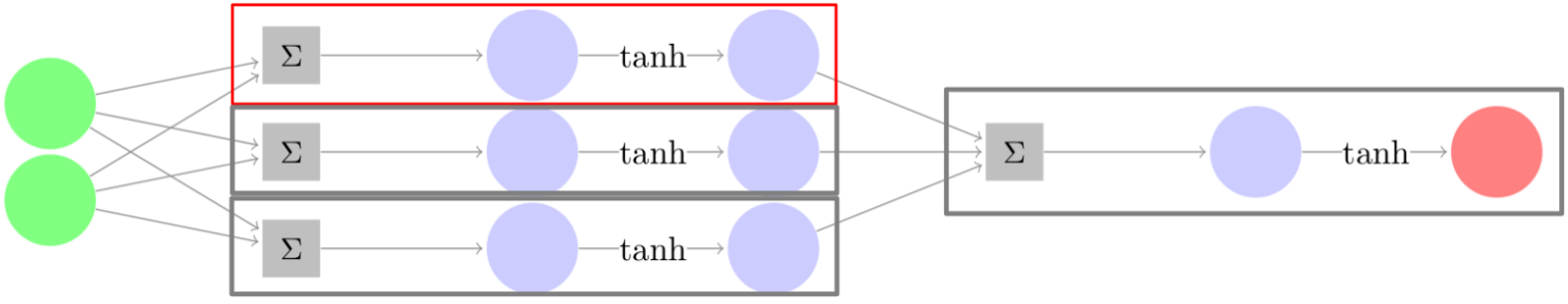
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Exkurs: Biologisches Neuron

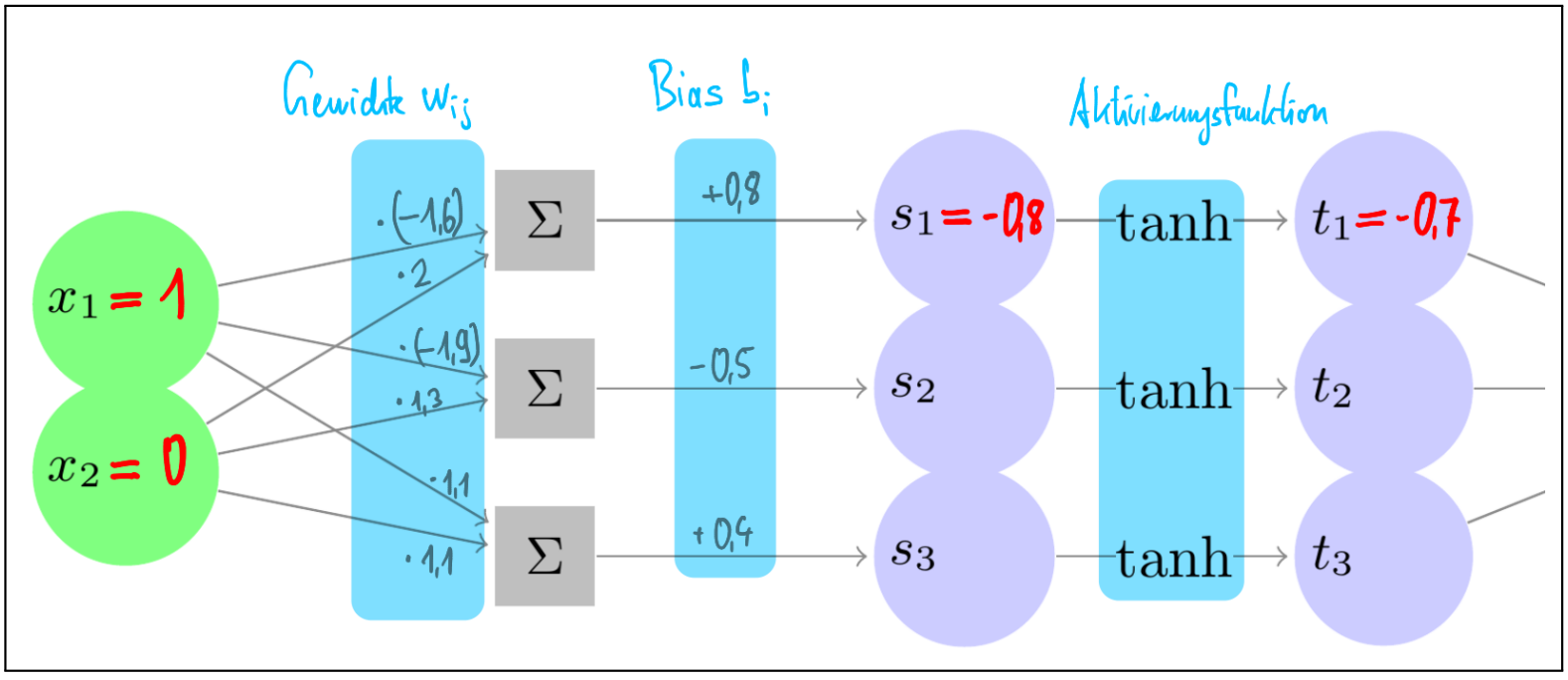


Unser künstliches neuronales Netz besteht aus vier Neuronen



Jedes Rechteck hat die Funktion eines Neurons

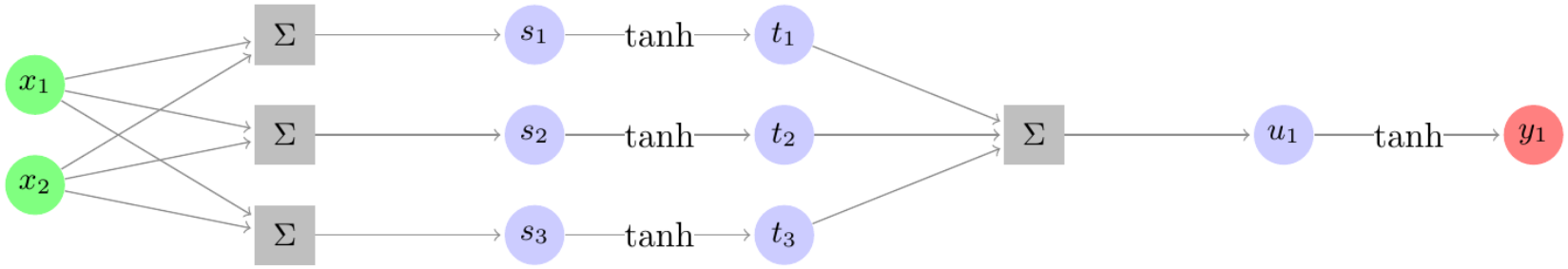
Beispiel eines neuronalen Netzes



Kompakte Notation durch Lineare Algebra

- $\boxed{s_1 = w_{11}x_1 + w_{12}x_2 + b_1}$
- Gewichte $W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$ und Bias $b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$
- $\boxed{s = Wx + b}$
- $\boxed{t = \tanh(Wx + b)}$

Beispiel eines neuronalen Netzes

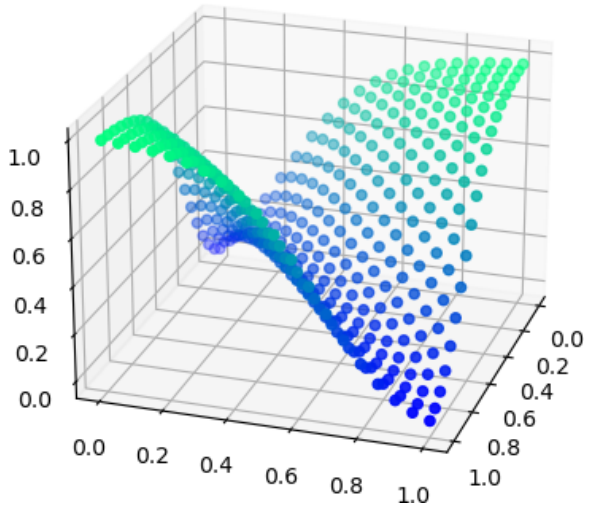


Neuronales Netz als Funktion

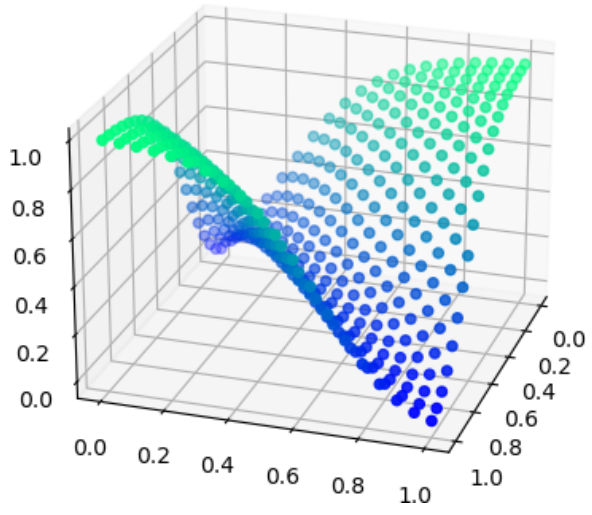
$$\mathbb{R}^2 \xrightarrow{(W,b)} \mathbb{R}^3 \xrightarrow{\tanh} \mathbb{R}^3 \xrightarrow{(W',b')} \mathbb{R} \xrightarrow{\tanh} \mathbb{R}$$

$$\boxed{y = \tanh(W' \tanh(Wx + b) + b')}$$

Zugehöriger Funktionsgraph



Beispiel eines neuronalen Netzes



Bemerkungen

- Neuronales Netz berechnet logisches XOR (= exklusives Oder = Entweder-Oder).
- Beispiel eigentlich langweilig, da XOR „durch Regeln“ definiert.
- ... aber schulgeeignet (inklusive tanh).
- nicht-lineare Aktivierungsfunktion (z.B. tanh) wichtig! Sonst erhalte nur affin-lineare Funktionen.

Entwurf neuronaler Netze als Curve Fitting (= Ausgleichsrechnung) Problem

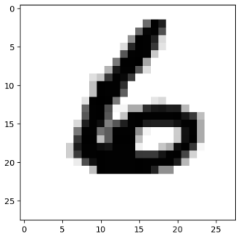
Gegeben Trainingsdaten (im Beispiel die Wahrheitstabelle von XOR):
Wie finde Gewichte und Bias so, dass die Trainingsdaten gut getroffen werden?

Netzarchitektur inklusive Aktivierungsfunktion per Ausprobieren und Erfahrung (Neuronale Netze entwerfen ist Ingenieurwissenschaft)

Programm demonstrieren

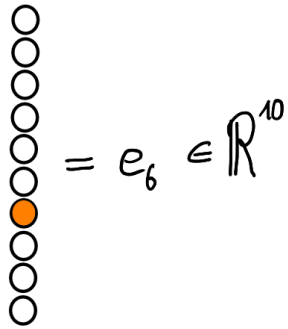
Ziffernerkennung – ein typisches Problem für neuronale Netze

Eingabe:
Ziffer aus MNIST-Trainingsdaten
28x28=784 Grauwerte



$\in \mathbb{R}^{784}$

Ausgabe:
Richtige LED soll leuchten



$= e_6 \in \mathbb{R}^{10}$

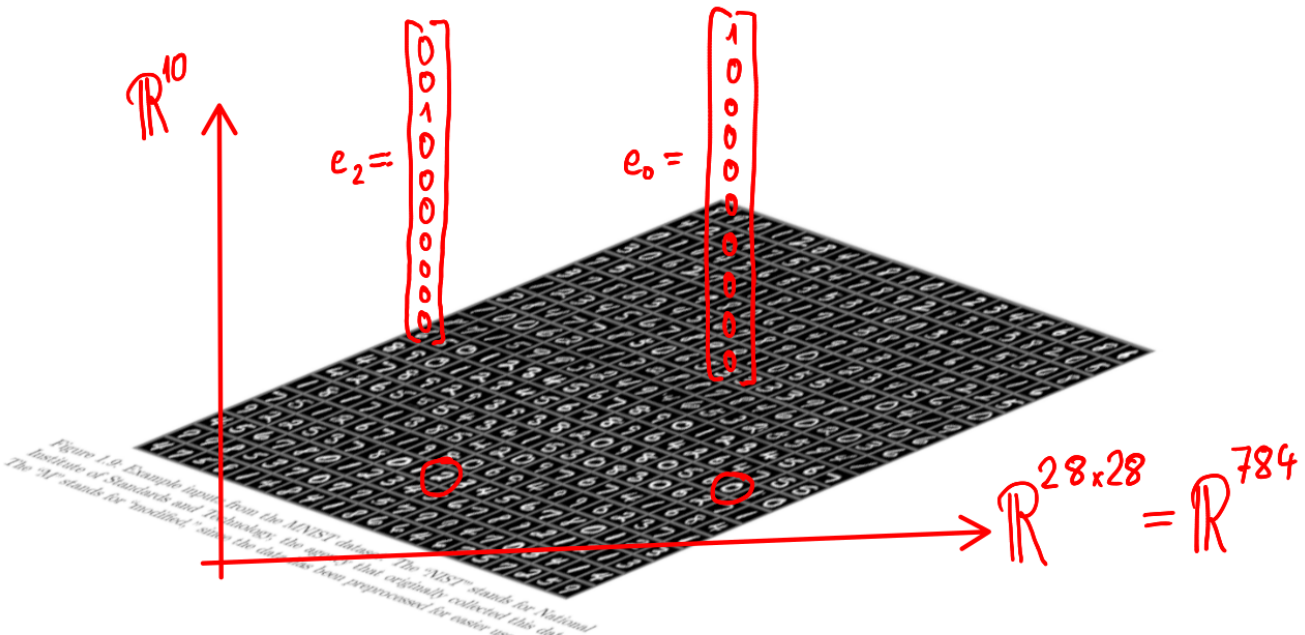
Die MNIST database ist ein Trainingsdatensatz aus 60'000 handschriftlichen Ziffern samt zugehöriger Ziffer.

Gesucht

Neuronales Netz zur Ziffernerkennung

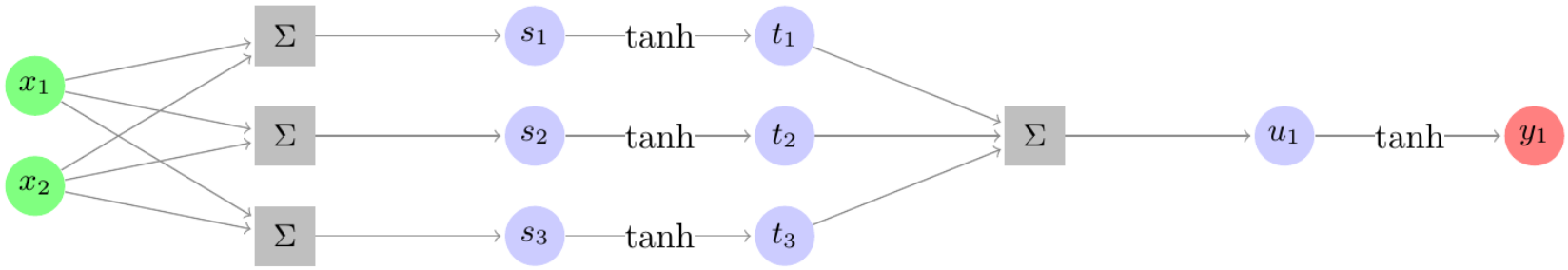
Mit anderen Worten: Approximiere die Trainingsdaten möglichst gut durch eine „Neuronale-Netz-Funktion“ (curve fitting).

$$\mathbb{R}^{784} \to \mathbb{R}^{10}$$



Netzentwurf (Ziffernerkennung)

Wir verwenden im Wesentlichen die Netzarchitektur des vorigen Beispiels



verwenden aber höherdimensionale Vektorräume:

$$\mathbb{R}^{784} \xrightarrow{(W,b)} \mathbb{R}^{40} \xrightarrow{\tanh} \mathbb{R}^{40} \xrightarrow{(W',b')} \mathbb{R}^{10} \xrightarrow{\tanh} \mathbb{R}^{10}$$

Gesucht sind möglichst gute

- Gewichte $W \in \mathbb{R}^{784\times 40} = \mathbb{R}^{31360}$ und $W' \in \mathbb{R}^{40\times 10} = \mathbb{R}^{400}$ und
- Bias $b \in \mathbb{R}^{40}$ und $b' \in \mathbb{R}^{10}$.

Lern- oder Trainingsphase (Ziffernerkennung)

Fixiere ein Trainingsdatum, bestehend aus

- dem Pixelbild einer handschriftlicher Ziffer $x \in \mathbb{R}^{784}$ und
- dem zugehörigen *Zielwert*, also der richtig erkannten Ziffer $z \in \mathbb{R}^{10}$ (aufgefasst als Einheitsvektor).

MSE = mean squared error = mittlere quadratische Abweichung

Wie weit ein beliebiges $y \in \mathbb{R}^{10}$ (alias eine Vorhersage des Netzes) vom Zielwert z entfernt ist, messen wir per

$$\mathrm{MSE}_z(y) := \frac{1}{10}\sum (y_i - z_i)^2 = \frac{1}{10}\,\|y - z\|^2$$

Rotationsparaboloid

Fehler von Netzen bei fixiertem Trainingsdatum (error/loss/cost function)

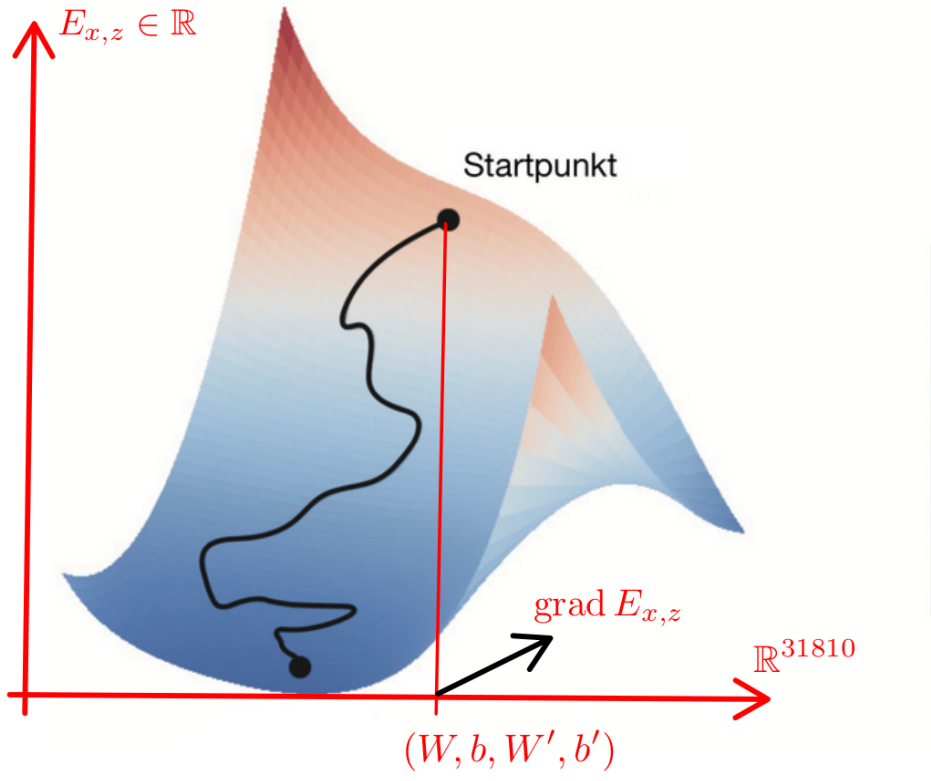
Betrachte die Zuordnung

$$E = E_{x,z} \colon \mathbb{R}^{31360+40+400+10} \longrightarrow \mathbb{R}$$

$$\underbrace{(W, b, W', b')}_{\substack{\text{Parameter eines} \\ \text{neuronalen Netzes}}} \mapsto \begin{pmatrix} \text{Vorhersage des zu-} \\ \text{gehörigen Netzes beim} \\ \text{(fixierten) Trainingsbild } x \end{pmatrix} \xmapsto{\mathrm{MSE}_z} \underbrace{E(W, b, W', b')}_{\substack{\text{Fehler beim} \\ \text{Trainingsbild } x}}$$

Sie misst für jedes (W, b, W', b'), wie falsch das zugehörige neuronale Netz beim fixierten Trainingsdatum (x, z) liegt.

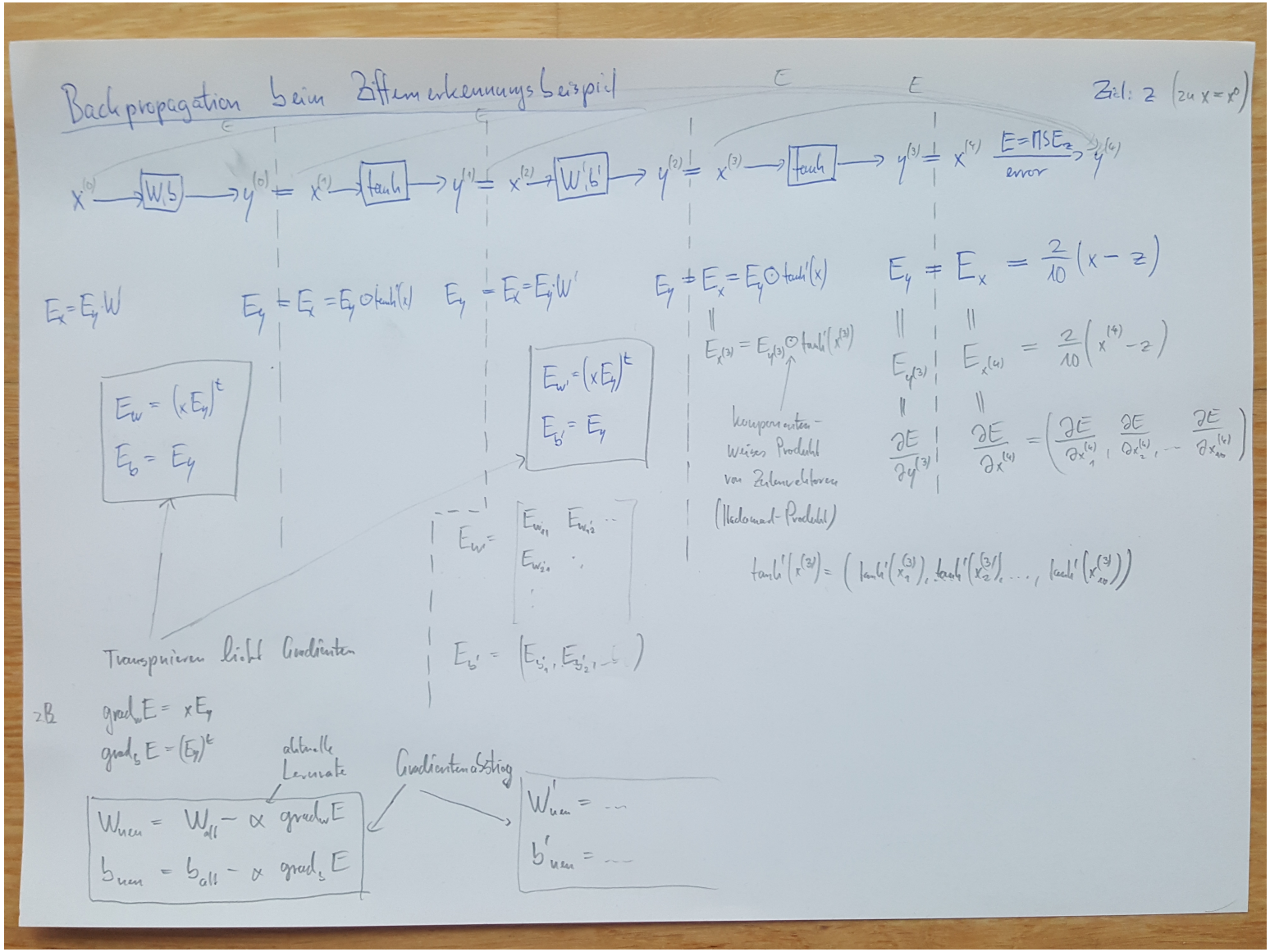
Lern- oder Trainingsphase: Gradientenabstieg und Backpropagation



- Bewege die Parameter (W, b, W', b') etwas in Richtung des negativen Gradienten von $E_{x,z}$ (= Richtung des lokal steilsten Abstiegs nach Cauchy-Schwartz), um den Fehler $E_{x,z}(W, b, W', b')$ zu verkleinern.
- Gradientenermittlung per Kettenregel: Backpropagation

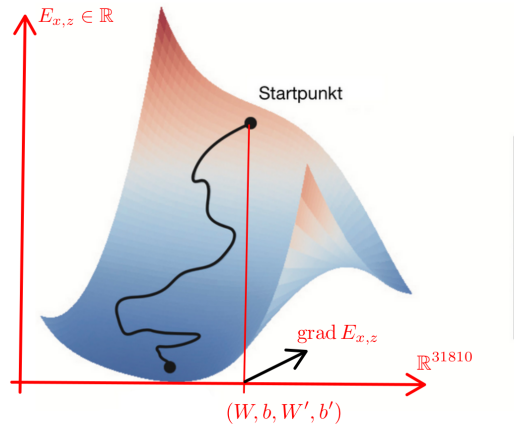
Mir scheint der Weg ins Tal in der Abbildung etwas kurvig.

Exkurs: Backpropagation im Detail



Die Bestimmung der Ableitungen ist startend bei $E_x = \frac{\partial E}{\partial x} = \frac{\partial E}{\partial x}(x) = \frac{2}{10}(x - z)$ von rechts zu lesen (backwards). Details zur Berechnung der Ableitung am Ende der Präsentation.

Trainingsalgorithmus (training loop)



Ab jetzt ist kein Trainingsdatum mehr fixiert.

Trainingsalgorithmus

1. Wähle zufällige (relativ kleine) Startwerte für W, W', b, b'.
2. Wiederhole mehrfach (ca. 10 Mal; jede Wiederholung heisst *Epoche*):
   - Gehe alle Trainingsdaten (x, z) durch:
     - Betrachte die zugehörige Fehlerfunktion $E_{x,z}$ und verändere (W, b, W', b') ein bisschen in Richtung des negativen Gradienten.

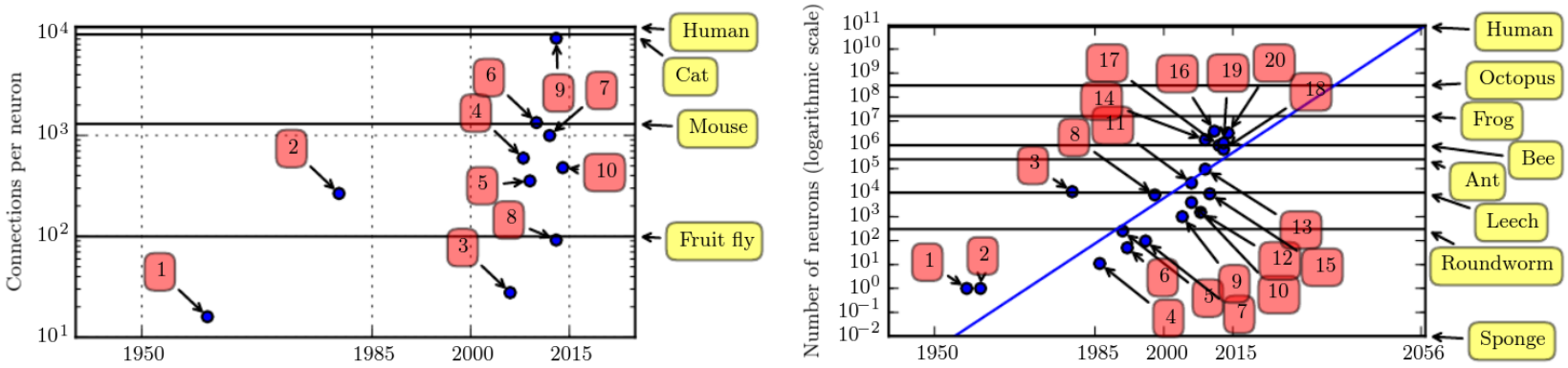
Am Anfang mag man sich vorstellen, dass nur ein Gewicht angepasst wird; dies könnte man auch in der Schule thematisieren.

Programme demonstrieren: Naives und besseres mit Keras und Tensorflow

Vergleich mit biologischen neuronalen Netzen

Motivation für neuronale Netze kommt aus der Biologie.

- biologische neuronale Netze (vor allem im Gehirn) sind nicht streng in Schichten eingeteilt, sondern viel wilder/chaotischer/zufälliger
- biologische Netze können viele verschiedene Aufgaben lösen, sind nicht so spezialisiert
- Lernalgorithmus vermutlich sehr verschieden (Gradientenabstieg?); deutlich weniger Trainingsdaten und Trainingszyklen nötig



Die roten Kästchen stehen für gewisse künstliche neuronale Netze.

Warum der KI-Hype jetzt?

Convolutional neural networks und *backpropagation*, die beiden Kernideen für *Computer Vision*[1], waren um 1990 gut verstanden.

Warum startete *deep learning* erst ab 2012 durch?

- bessere Hardware (so dass sehr grosse Netze verwendet werden können)
  - high performance chips für Gaming (GPUs = graphical processing units)
  - TPUs = tensor processing units[2]; im Jahr 2020: TPU 10'000 Mal so schnell wie bester Supercomputer 1990
  - aktuell wohl Nvidia führend
  - Fortschritte bei Speicher-Medien
- viele immer bessere Daten
  - z.B. massive Expansion des Internets, viele Testdaten verfügbar (Wikipedia, YouTube, Flickr)
- algorithmische Fortschritte (ausprobierbar dank obiger Punkte)
  - z.B.Transformer Architektur seit 2017: Revolutionär bei der Verarbeitung natürlicher Sprache (etwa in ChatGPT oder vermutlich hier)

Weitere Faktoren:

- viel Geld von Investoren
  - 2017 ca. 16 Milliarden US$ für KI start ups weltweit, 2011 ca. 1 Milliarde US$
  - GAFAM (Google/Alphabet, Apple, Facebook/Meta, Amazon (AWS), Microsoft) etc. geben intern riesige Summen aus
- Demokratisierung von deep learning:
  - Viele leicht zugängliche Bibliotheken: TensorFlow, Keras, PyTorch, (wohl recht neu: Google Jax)

[1] Hauptprobleme: Bildklassifizierung, Bildsegmentierung, Objektdetektion

[2] Tensor wird als synonym für mehrdimensionale Matrizen verwendet; ein 0-D-Tensor ist ein Skalar, ein 1-D-Tensor ein Vektor, ein 2-D-Tensor eine Matrix, etc.

Word embeddings (nützlich für alle texterzeugende KI)

Wie speichert man Wörter als Vektoren?

- Naiv: z.B. 20'000 Wörter, jedes Wort ist Standard-Basisvektor im $\mathbb{R}^{20000}$
- Besser: word embeddings: niedrigdimensionaler Vektorraum (etwa $\mathbb{R}^{1024}$); semantische Beziehung zwischen Wörtern sollte geometrisch sichtbar sein:

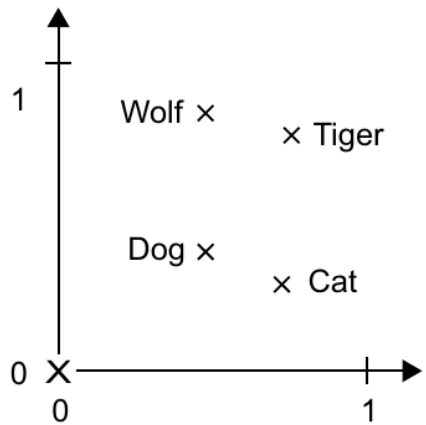


Figure 11.3 A toy example of a word-embedding space

  - Haustier-zu-wildem-Tier-Vektor
  - Hundeartig-zu-katzenartig-Vektor
  - usw., etwa Männlich-zu-weiblich-Vektor (Mann zu Frau, König zu Königin)

Vgl.: Word2vec
Schulgeeignet?

Einige interessante Punkte aus dem letzen Kapitel des Buchs von Chollet

Intelligenz: Je intelligenter, desto weniger Vorwissen bei neuen Aufgaben nötig.
Die heutige AI sollte besser *artificial cognition* heissen.

Abstraktion (Fähigkeit dazu Teil von Intelligenz) hat zwei Extreme:

- exact structural match: program/structure-centric analogies
  - Z.B. mehrfache Verwendung derselben Funktion beim Programmieren; Erkennen isomorpher Strukturen (etwa Definition eines topologischen Raums)
  - liegt dem Argumentieren und Planen zu Grunde
  - wenige Beispiele reichen
  - deep learning schlecht:
    - 5 Zahlen sortieren per neuronalem Netz geht, aber erstaunlich schwierig; Verallgemeinerung auf 10 Zahlen: neu trainieren.
    - Aber: kurzes Python-Programm, funktioniert auch für 1'000'000 Zahlen
- similarity comparison: value-centric analogies
  - Z.B: Ziffernerkennung
  - liegt der Wahrnehmung und Intuition zu Grunde
  - benötigt viel Erfahrung
  - deep learning gut darin
    - Ziffernerkennung gut und einfach per deep learning lösbar
    - Von Hand: sehr aufwändig

Was bringt die Zukunft?

Vielleicht komplexere Modelle: Nicht nur Gewichte und Bias und diverse Architekturen, sondern auch Kontrollstrukturen wie in Computerprogrammen?

Literatur und andere Quellen

- François Chollet: Deep Learning with Python. Manning Publications, 2021.
  https://www.manning.com/books/deep-learning-with-python
- Ian Goodfellow, Yoshua Bengio, Aaron Courville. MIT Press, 2016.
  https://www.deeplearningbook.org/
- Pit Noack, Sophia Sanner: Künstliche Intelligenz verstehen, eine spielerische Einführung in KI. Rheinwerk-Verlag, 2023.
  Homepage und Quellcodes:
  https://www.rheinwerk-verlag.de/kuenstliche-intelligenz-verstehen-eine-spielerische-einfuehrung-in-die-ki/
  Online Ausprobiermaterial: https://www.maschinennah.de/ki-buch/
- Omar Aflak on YouTube: The Independent Code: Neural Network from Scratch, Mathematics & Python Code
  https://www.youtube.com/watch?v=pauPCy_s0Ok
  Artikel mit fast demselben Inhalt:
  https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65

Bemerkungen

- Buch von François Chollet (Schöpfer der Keras-Bibliothek):
  Mit Hilfe von Keras und Tensorflow lernt man, neuronale Netze in wenigen Zeilen in Python zu programmieren; viele praktische Beispiele werden implementiert (Ziffernerkennung, Übersetzung, DeepDream-artige Bilderzeugung).
  Theorie und Geschichtliches werden erklärt.
  Auch Philosophisches: Was ist Intelligenz? Welche Arten gibt es? Zukunft von KI?
- Buch von Pit Noack (inklusive Online-Material zum Ausprobieren):
  Zum allgemeinen Einstieg in KI besonders geeignet, auch für Laien.
- Video und Artikel von Omar Aflak:
  Sehr empfehlenswert. Theoretisch und praktisch (in sehr elegantem objektorientiertem Python) werden neuronale Netze erklärt. Ich kann mir kaum einen besseren und schnelleren Zugang vorstellen. Man sollte mit Python vertraut sein und keine Angst vor partiellen Ableitungen haben.

  Manche Ableitungen würde ich eher als Zeilen- statt Spaltenvektoren schreiben.

Ende

Anhang

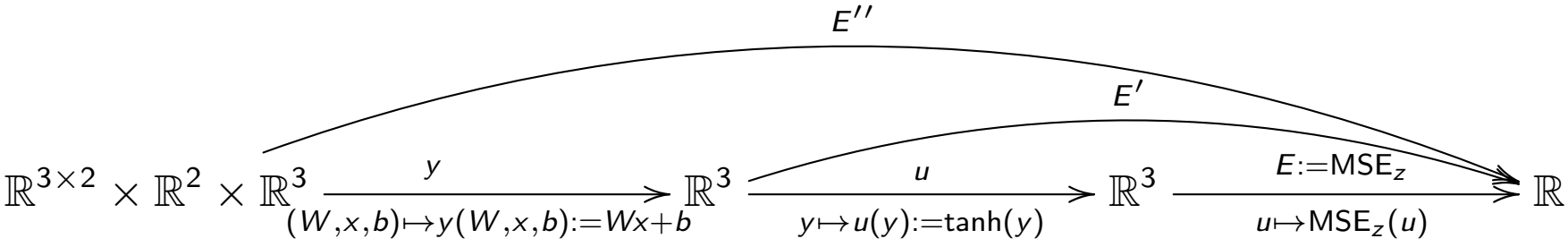
Auf den folgenden Seiten befinden sich einige Notizen zu Backpropagation, die mir beim Verstehen geholfen haben.
Achtung: Ich habe sie ziemlich rasch aufgeschrieben; sie sind (noch) nicht gut aufbereitet.
Ich empfehle, parallel https://www.youtube.com/watch?v=pauPCy_s0Ok oder das Foto oben zu Backpropagation (Benennung der Variablen unterschiedlich) anzuschauen.

Backpropagation: Die Formeln an einem aussagekräftigen Beispiel

Struktur des betrachteten neuronalen Netzes:

$$\mathbb{R}^2 \xrightarrow{(W,b)} \mathbb{R}^3 \xrightarrow{\tanh} \mathbb{R}^3$$

Trainingsdatum (x, z) fixiert. Drei Fehlerfunktionen E, E', E''.



Nun berechnen wir die Ableitungen (und damit Gradienten) der Fehlerfunktionen E'', E' und E und beginnen rechts, um uns dann per Kettenregel nach links voranzutasten (backpropagation).
Ableitung von E, also bei MSE-Schicht
Ableitung von E.

$$E_u(u) = \frac{\partial E}{\partial u}(u) := \begin{bmatrix} \frac{\partial E}{\partial u_1}(u) & \frac{\partial E}{\partial u_2}(u) & \frac{\partial E}{\partial u_3}(u) \end{bmatrix}$$
$$= \frac{2}{3}\begin{bmatrix} u_1 - z_1 & u_2 - z_2 & u_3 - z_3 \end{bmatrix} = \frac{2}{3}(u - z)$$

Ableitung bei Aktivierungsschicht

Ziel: Ableitung von E'.

Ableitung von $u = u(y)$:

$$u_y(y) = \frac{\partial u}{\partial y}(y) := \begin{bmatrix} \frac{\partial u_1}{\partial y_1} & \frac{\partial u_1}{\partial y_2} & \frac{\partial u_1}{\partial y_3} \\ \frac{\partial u_2}{\partial y_1} & \frac{\partial u_2}{\partial y_2} & \frac{\partial u_2}{\partial y_3} \\ \frac{\partial u_3}{\partial y_1} & \frac{\partial u_3}{\partial y_2} & \frac{\partial u_3}{\partial y_3} \end{bmatrix} (y) = \begin{bmatrix} \tanh'(y_1) & 0 & 0 \\ 0 & \tanh'(y_2) & 0 \\ 0 & 0 & \tanh'(y_3) \end{bmatrix}$$

Induktiv nimmt man an, dass der Zeilenvektor $\frac{\partial E}{\partial u}(u(y))$ bereits berechnet ist.

Also

$$E'_y(y) = \frac{\partial E'}{\partial y}(y) = \frac{\partial (E \circ u)}{\partial y}(y) = \underbrace{\frac{\partial E}{\partial u}(u(y))}_{\text{bereits berechnet, im konkreten Beispiel könnte einsetzen, aber für allgemeinen Fall eher irreführend.}} \cdot \frac{\partial u}{\partial y}(y)$$

Das unterklammerte ist ein Zeilenvektor. Seine i-te Komponenten wird mit $\tanh'(y_i)$ multipliziert (wegen der Diagonalmatrix oben).

Ableitung bei Gewichts-Bias-Schicht (= affin-linearer Schicht)

Ziel: Ableitung von E''.

Ableitung von $y = y(x)$:

$$y_{(W,x,b)}(W,x,b) = \begin{bmatrix} \frac{\partial y}{\partial W} & \frac{\partial y}{\partial x} & \frac{\partial y}{\partial b} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial y_1}{\partial W_{11}} & \frac{\partial y_1}{\partial W_{12}} & \frac{\partial y_1}{\partial W_{21}} & \frac{\partial y_1}{\partial W_{22}} & \frac{\partial y_1}{\partial W_{31}} & \frac{\partial y_1}{\partial W_{32}} & \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \frac{\partial y_1}{\partial b_1} & \frac{\partial y_1}{\partial b_2} & \frac{\partial y_1}{\partial b_3} \\ \frac{\partial y_2}{\partial W_{11}} & \cdots & & & & & & & & & \\ \frac{\partial y_3}{\partial W_{11}} & \cdots & & & & & & & & & \end{bmatrix}$$

$$= \begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 & w_{11} & w_{12} & 1 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 & w_{21} & w_{22} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 & w_{31} & w_{32} & 0 & 0 & 1 \end{bmatrix}$$

wohl nicht nötig: Wenn man diese lineare Abbildung (= Ableitung) auf einen Tangentialvektor (W', x', b') loslässt, bekommt man (im Kopf (W', x', b') als Spaltenvektor schreiben, der der transponierte von $w'_{11}, w'_{12}, w'_{21}, \ldots x'_1, x'_2, b'_1, b'_2, b'_3$ ist)

$$W'x + Wx' + b'$$

Induktiv nimmt man nun an, dass der Zeilenvektor $\frac{\partial E'}{\partial y}(y(x)) = \frac{\partial E'}{\partial y}(Wx+b)$ bereits berechnet ist. Wir rechnen nun die drei partiellen Ableitungen von E nach W, nach b und (obwohl x in der Mitte steht) am Ende nach x separat aus.
Zuerst nach W:

$$\frac{\partial E''}{\partial W}(W,x,b) = \frac{\partial (E' \circ y)}{\partial W}(W,x,b) = \underbrace{\frac{\partial E'}{\partial y}(Wx+b)}_{\text{bereits berechnet}} \cdot \underbrace{\begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{bmatrix}}_{\frac{\partial y}{\partial W}(W,x,b)}$$

$$= \begin{bmatrix} \frac{\partial E'}{\partial y_1} & \frac{\partial E'}{\partial y_2} & \frac{\partial E'}{\partial y_3} \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial E'}{\partial y_1} x_1 & \frac{\partial E'}{\partial y_1} x_2 & \frac{\partial E'}{\partial y_2} x_1 & \frac{\partial E'}{\partial y_2} x_2 & \frac{\partial E'}{\partial y_3} x_1 & \frac{\partial E'}{\partial y_3} x_2 \end{bmatrix}$$

$$\overset{\text{als Matrix}}{=} \begin{bmatrix} \frac{\partial E'}{\partial y_1} x_1 & \frac{\partial E'}{\partial y_1} x_2 \\ \frac{\partial E'}{\partial y_2} x_1 & \frac{\partial E'}{\partial y_2} x_2 \\ \frac{\partial E'}{\partial y_3} x_1 & \frac{\partial E'}{\partial y_3} x_2 \end{bmatrix} = \left(\frac{\partial E'}{\partial y}\right)^{\mathrm{t}} \cdot x^{\mathrm{t}} = \boxed{\left(x \cdot \frac{\partial E'}{\partial y}\right)^{\mathrm{t}}}$$

und wer will kann dies wieder als Zeilenvektor schreiben...

Nach b ableiten ist langweilig:

$$\frac{\partial E''}{\partial b}(W,x,b) = \frac{\partial (E' \circ y)}{\partial b}(W,x,b) = \underbrace{\frac{\partial E'}{\partial y}(Wx+b)}_{\text{bereits berechnet}} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\frac{\partial y}{\partial b}(W,x,b)} = \frac{\partial E'}{\partial y}(Wx+b)$$

Nun noch nach x ableiten, denn dies dient im Allgemeinen der nächsten Schicht als Fehlerableitung bei der Backpropagation:

$$\frac{\partial E''}{\partial x}(W,x,b) = \frac{\partial (E' \circ y)}{\partial x}(W,x,b) = \underbrace{\frac{\partial E'}{\partial x}(Wx+b)}_{\text{bereits berechnet}} \cdot \underbrace{\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}}_{\frac{\partial y}{\partial x}(W,x,b)}$$

$$= \frac{\partial E'}{\partial y}(Wx+b) \cdot W$$

Zusammenfassung affin-lineare Schicht

Die drei gerade berechneten Formeln für Backpropagation bei einer „affin-linearen Schicht“ $x \mapsto Wx + b$ kann man kurz so aufschreiben:

$$\frac{\partial E''}{\partial W} = \left(x \cdot \frac{\partial E'}{\partial y} \right)^{\mathrm{t}}$$
$$\frac{\partial E''}{\partial b} = \frac{\partial E'}{\partial y}$$
$$\frac{\partial E''}{\partial x} = \frac{\partial E'}{\partial y} \cdot W$$

(Gradienten per Transponieren) Mit anderen Worten: Ist der Gradient bei y bekannt, so ist er auch bei (W, x, b) bekannt und man kann W und b korrigieren. Den Gradient bei x braucht man fürs iterierte Backpropagieren. Alternativnotation der obigen Formeln:

$$E''_W = \left(x \cdot E'_y \right)^{\mathrm{t}}$$
$$E''_b = E'_y$$
$$E''_x = E'_y \cdot W$$