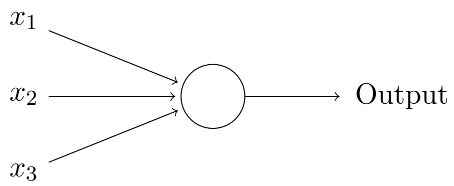
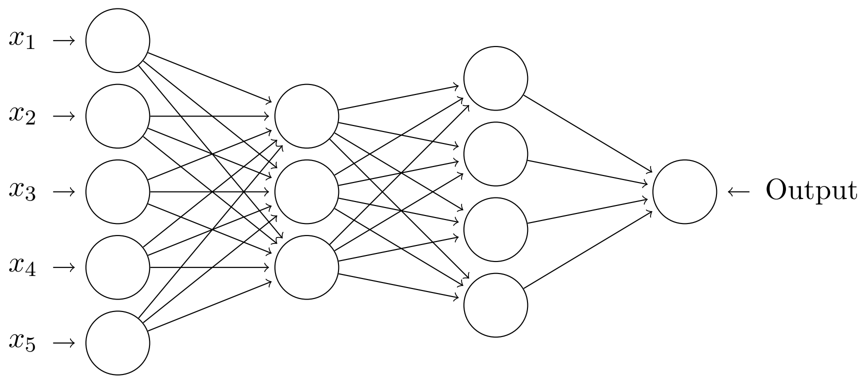
1 Perceptrons

- Modell eines künstlichen Neurons
- Vorgänger der *Sigmoid Neurons*, die in heutigen modernen neuronalen Netzen benutzt werden



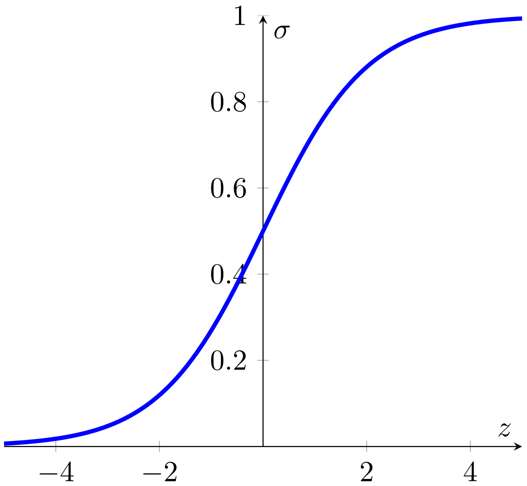
- Eingaben: $x_1, x_2, \ldots, x_n \in \{0, 1\}$
- *Weights*: $w_1, w_2, \ldots, w_n \in \mathbb{R}$ für jede Eingabe x_1, der die jeweilige Eingabe gewichtet
- Output $= \begin{cases} 0, & \text{wenn } \sum_j w_j x_j \leq \text{Treshold, wobei Treshold} \in \mathbb{R} \\ 1, & \text{sonst} \end{cases}$
- **vereinfachte Schreibweise:**
  - $\sum_j w_j x_j = w \cdot x$, wobei w und x nun Vektoren beschreiben, dessen Komponenten die Gewichte und Eingaben sind
  - ziehe den Treshold auf die andere Seite der Ungleichung (*Bias*: $b = -$Treshold)
  - $\Rightarrow$ Bias beschreibt, wie einfach es ist ein Perceptron auf 1 zu bringen
  - Output $= \begin{cases} 0, & \text{wenn } w \cdot x + b \leq 0 \\ 1, & \text{sonst} \end{cases}$
- Mit Hilfe eines *neuronalen Netzes* aus Perceptons können kompliziertere Entscheidungen getroffen werden:



- neuronales Netz besteht aus drei Schichten: *Input-Layer*, *Hidden-Layer* und *Output-Layer*
- Ziel: bringe das Netz dazu zu lernen, d.h. ihre Weights und Bias-Werte anzupassen, sodass für jede Eingabe das erwartete Ergebnis erzielt wird

2 Sigmoid Neurons

- Anforderung: Eine kleine Änderung in den Weights/Bias-Werten führt nur zu einer kleinen Änderung in der Ausgabe
- $\Rightarrow$ Perceptrons sind dafür nicht geeignet, da sie nur flippen können
- *Sigmoid Neurons*:
  - Eingaben: $x_1, x_2, \ldots, x_n \in [0, 1]$
  - Ausgabe: $\sigma(w \cdot x + b) = \sigma(z) = \frac{1}{1+e^{-z}}$
  - Ähnlichkeit: $z \to \infty \Rightarrow \sigma(z) \approx 1$ und $z \to -\infty \Rightarrow \sigma(z) \approx 0$



3 Neuronale Netze

- **Beispiel:** Schrifterkennung
  - Eingabe: 28 Pixel $\times$ 28 Pixel = 784 Neuronen mit Intensität $\in [0, 1]$
  - Ausgabe: 10 Neuronen, die die Wahrscheinlichkeiten beschreiben, dass das Bild die entsprechende Zahl zeigt, d.h. Output > 0.5
  - Ziel: approximiere die Funktion $y(x)$, die die Trainingdaten beschreibt, d.h. $y(x) = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$ für ein Bild x mit einer Null, usw.
  - Methode: minimiere Kostenfunktion $C(w, b) = \frac{1}{2n} \sum_x ||y(x) - a(x)||^2$, wobei $a(x)$ der Output des neuronalen Netzes bei Input x ist

3.1 Gradient descent

- Ziel: Lösung des Minimierungsproblems $C(v) = C(v_1, v_2, \ldots, v_n)$
- Lösung:
  - definiere den Gradienten $\nabla C = (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, \ldots, \frac{\partial C}{\partial v_n})^T$ von C aus den partiellen Ableitungen $\frac{\partial C}{\partial v_i}$ von C

- $\Rightarrow \Delta C \approx \nabla C \cdot \Delta v$
- wähle $\Delta v = -\eta \nabla C$, dann gilt $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2$
  - η wird das *Lerntempo* (engl. *learning rate*) genannt
  - da $||\nabla C||^2 \geq 0$ folgt, dass $\Delta C \leq 0$ für alle $\eta \in \mathbb{R}^+$
  - wähle η so, dass wir nicht zu langsam lernen, aber dennoch eine gute Approximation erhalten

3.1.1 Gradient descent in neuronalen Netzen

- ∇C besteht aus partiellen Ableitungen der Komponenten w_k und b_l
- Problem: ∇C wird berechnet aus dem Mittelwert der Gradienten ∇C_x für alle Trainingsdaten $x \Rightarrow$ berechnungsintensiv
- Lösung: *Stochastic gradient descent*
  - Idee: berechne ∇C aus einer kleinen Anzahl m zufällig gewählter Trainingsdaten $X_1, X_2, \ldots, X_m$
  - vorausgesetzt m ist groß genug, dann gilt $\frac{1}{m} \sum_{i=1}^{m} \nabla C_{X_i} \approx \frac{1}{n} \sum_x \nabla C_x = \nabla C$

4 Backpropagation