Machine Learning

Deutsch - Maschinelles Lernen: Wie ein KI-Modell aus Daten Muster lernt, um aus neuen Daten Vorhersagen zu treffen (Inferenz). Im Wesentlichen gibt es drei Arten.

- Supervised Learning (Überwachtes ML)
- Unsupervised Learning (Unüberwachtes ML)
- Reinforcement Learning (Verstärkungslernen)

In dieser Sitzung werden wir über Supervised Learning sprechen, mit einem besonderen Schwerpunkt auf **Artificial Neural Networks (ANNs)**.

Supervised Learning

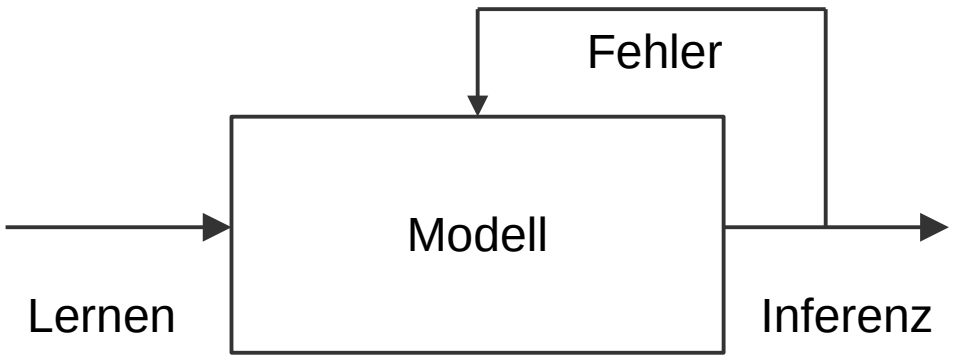
Der Algorithmus wird auf einem beschrifteten Datensatz trainiert, d. h. jede Eingabe hat eine bekannte Ausgabe (Beschriftung).

Beispiel: Ein Datensatz zum Trainieren eines Modells zur Erkennung von Spam-E-Mails würde mehrere *Inputs* für die Betreffzeile enthalten, wobei die korrekte *Outputs* als Spam oder Nicht-Spam gekennzeichnet wird.

Betreff	**Label**
“Ihr Konto wurde eingeschränkt”	Spam
“Sie haben im Lotto gewonnen!”	Spam
“Teamtreffen um 15:00 in Raum 201”	Nicht-Spam

Modell-Lern-Inferenz

In unserem Modell-Lern-Inferenz-Paradigma wird überwachtes Lernen durchgeführt, indem Dateneingaben in das Modell eingespeist werden, seine Vorhersagen mit bekannten korrekten Ausgaben verglichen werden und das Modell dann auf der Grundlage des gemessenen Fehlers angepasst wird.



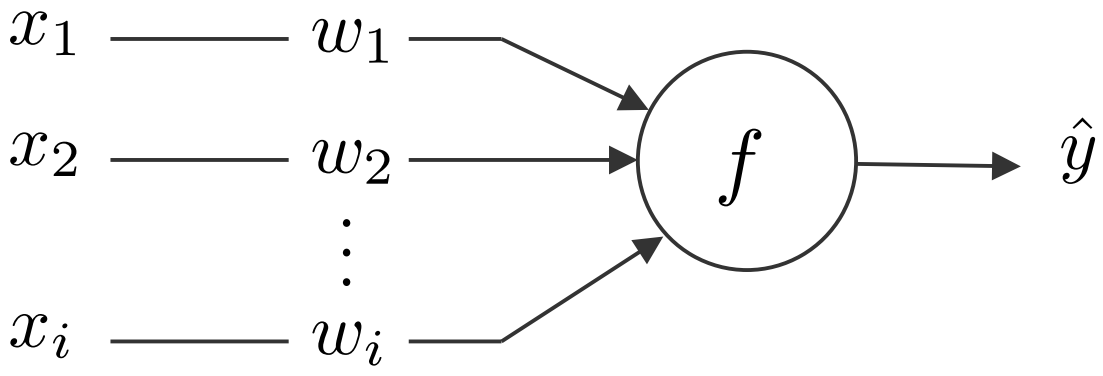
Arten von Problemen

Zwei Arten von Problemen werden hier behandelt: **Classification** (Klassifizierung) und **Regression**. Der Unterschied liegt in der Art der Ausgabe, die das Modell vorhersagt:

	Classification	**Regression**
Output Type	Klasse/Kategorie	Reelle Zahl
Example	Spam (0 oder 1)	12.6 (Grad)

Perceptron

Bei ANNs wird das Supervised Learning erreicht, indem Daten in die Eingänge des Netzes eingespeist werden und daraus eine Ausgabe abgeleitet wird. Der Fehler in der Vorhersage wird berechnet und als Parameter für die Anpassung der Gewichte des Modells verwendet. Das einfachste ANN ist das einzelne Perceptron.



Perceptron

Bei einem Perzeptron ist die Ausgabe:

$$\hat{y} = f(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$$

Mit:

- Skalarprodukt aus dem Gewichtsvektor und dem Inputvektor $\mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Bias-Parameter b
- Aktivierungsfunktion f

Perceptron - Classification

Perceptrons sind ideal für Klassifizierungsprobleme, bei denen die Klassen **linear trennbar** sind. Ein binäres Klassifizierungsproblem mit einem Eingangsvektor $\mathbf{x}$ und einem Label y ist linear trennbar, wenn es $\mathbf{w}$ und b gibt, dass:

$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b > 0$$

und $$\hat{y} = 0 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b < 0$$

Beispiel – Autonomes Auto

Im Beispiel des autonomen Autos gibt es zwei binäre Sensoren - einen Fußgängersensor **F** und einen Sensor für eine rote Ampel **R**. Wenn ein Fußgänger vor das Auto tritt, ist die Anhaltebedingung **S** erfüllt. Wenn eine rote Ampel erkannt wird, ist die Anhaltebedingung erfüllt. Wenn sowohl ein Fußgänger als auch eine rote Ampel erkannt werden, ist die Anhaltebedingung erfüllt. In allen anderen Fällen ist die Anhaltebedingung falsch.

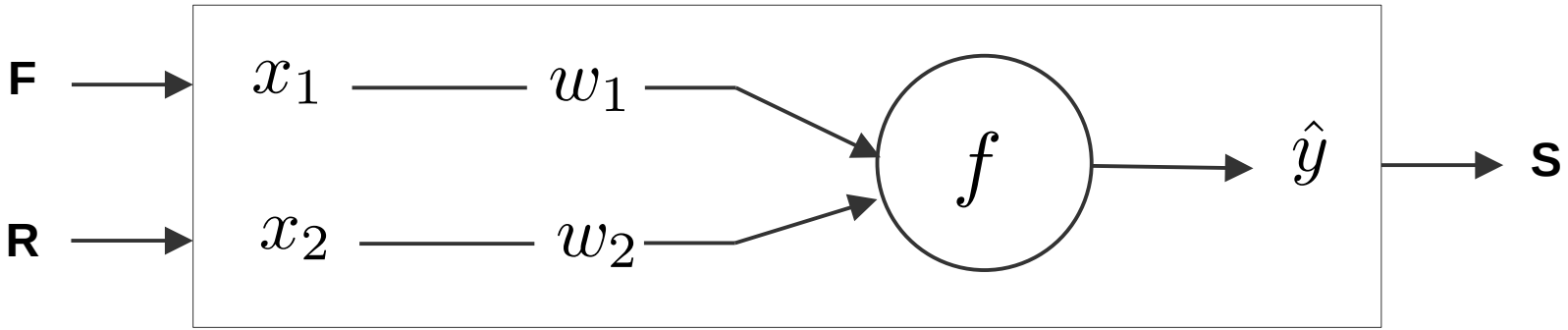
F	**R**	**S**
0	0	0
0	1	1
1	0	1
1	1	1

Dies ist eine ODER-Funktion, oder in der Logik:

$$(F \vee R) \rightarrow S$$

Beispiel – Autonomes Auto

Diese Funktion kann mit einem einzigen Perzeptron implementiert werden. Das Perceptron-Modell ist unten dargestellt, wobei der Eingangsvektor $\mathbf{x}$ die Sensoren **F** und **R** und der Ausgang y die Stoppbedingung **S** darstellt.



Beispiel – Autonomes Auto

- Das binäre Klassifizierungsproblem ist *linear trennbar*, da für die folgenden gewählten Werte $w_1 = 1$, $w_2 = 1$, und $b = -0.5$, gilt:

F	R	S
0	0	0
0	1	1
1	0	1
1	1	1

$$\hat{y} = 0 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0 \cdot 1 + 0 \cdot 1 + (-0.5) = -0.5 < 0$$

$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0 \cdot 1 + 1 \cdot 1 + (-0.5) = 0.5 > 0$$

$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 1 \cdot 1 + 0 \cdot 1 + (-0.5) = 0.5 > 0$$

$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 1 \cdot 1 + 1 \cdot 1 + (-0.5) = 1.5 > 0$$

Perceptron - Regression

Für Regressionsprobleme sind Perceptrons nur dann nützlich, wenn die zu modellierenden Systeme linear sind. Ein **lineares System** S ist ein System, in dem das Prinzip der **Superposition** gilt:

$$\alpha S(\mathbf{x}) = S(\alpha \mathbf{x})$$

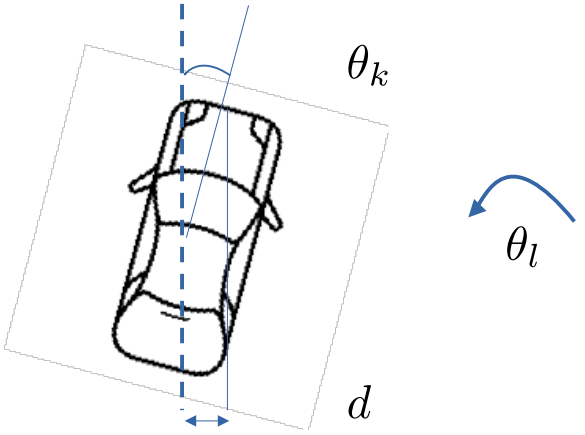
und $$S(x_1 + x_2) = S(x_1) + S(x_2)$$

Wo α ein Skalar ist.

Beispiel – Autonomes Auto

In diesem Beispiel hat die selbstfahrenden Auto eine Kurskorrektur, wobei der aktuelle Kurs θ_k und der Abstand d der Vorderseite der Auto von der Mittellinie die Eingaben x_1 und x_2 sind. Die Ausgabe $\hat{y}$ ist der Lenkwinkel, θ_l der zur Korrektur des Kurses erforderlich ist.

$x_1 = \theta_k$	$x_2 = d$	$\hat{y} = \theta_l$
-2	+0.4	-10
-1	+0.2	-5
+2	-0.3	+9
-4	+0.8	-20



Beispiel – Autonomes Auto

Das System kann wie folgt beschrieben werden:

$$\hat{y} = S(\mathbf{x}) \qquad \text{wo:} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Wir sehen ob das gilt:

$$\alpha S(\mathbf{x}) = S(\alpha \mathbf{x})$$

Prüfung der Homogenität

Wenn die Eingänge und Ausgänge skalieren, weist das System **Homogenität** auf und ist wahrscheinlich linear! Dies kann getestet werden, indem man eine Zeile der Daten durch eine andere dividiert und einen Wert für α erhält.

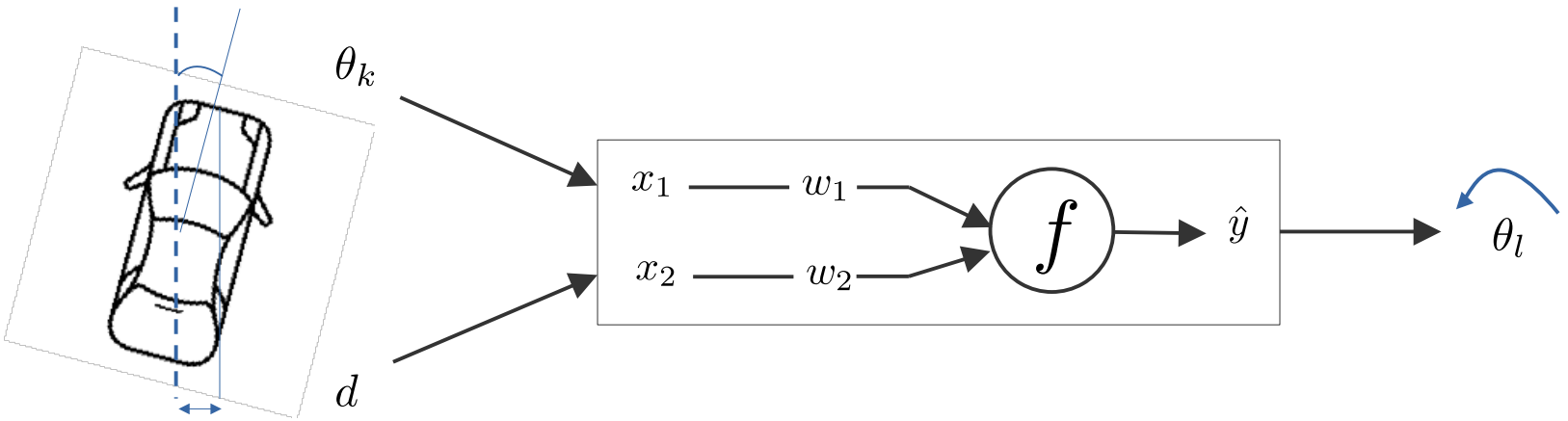
$x_1 = \theta_k$	$x_2 = d$	$\hat{y} = \theta_l$
-2	+0.4	-10
-1	+0.2	-5
+2	-0.3	+9
-4	+0.8	-20

$$\alpha\mathbf{x} = \begin{pmatrix} \frac{-2}{-1}x_1 \\ \frac{+0.4}{+0.2}x_2 \end{pmatrix} = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix}$$

$$\mathrm{S}(\alpha\mathbf{x}) = \frac{-10}{-5}\hat{y} = 2\hat{y}$$

Beispiel – Autonomes Auto

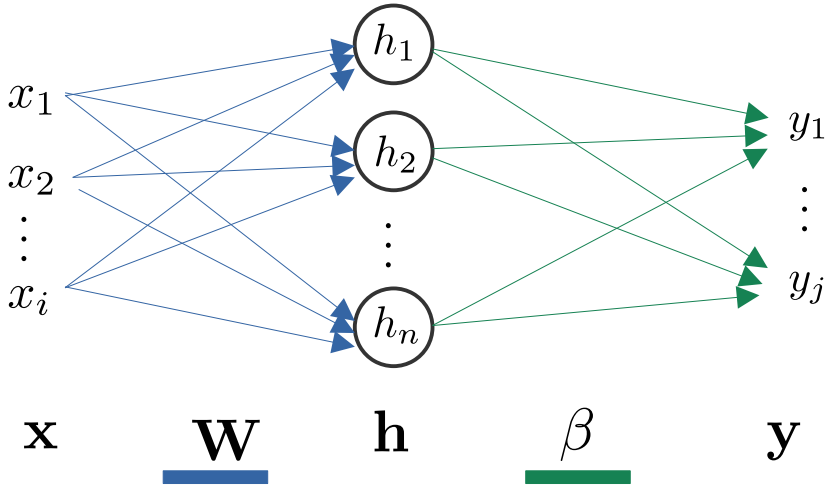
Das Steuerungssystem für das Auto könnte also als Perceptron implementiert werden.



Multilayer Perceptron

Bei Daten für ein System, das **nichtlineare** Merkmale aufweist - d. h. für einen Klassifikator mit nichtlinear trennbaren Klassen oder ein System mit nichtlinearer Regression - ist ein Perceptron nicht in der Lage, die durch das System beschriebene Funktion zu approximieren. In diesem Fall ist ein Multilayer Perceptron erforderlich. „Multilayer“ (Mehrsichtig) bedeutet, dass sich zwischen der Eingabe- und der Ausgabeschicht eine oder mehrere verborgene Schichten befinden.

Multilayer Perceptron (MLP)



$$\mathbf{x} \in \mathbb{R}^{d_{in}}$$
$$\mathbf{W} \in \mathbb{R}^{d_{in} \times d_h}$$
$$\mathbf{h} \in \mathbb{R}^{d_h}$$
$$\beta \in \mathbb{R}^{d_h \times d_{out}}$$
$$\mathbf{y} \in \mathbb{R}^{d_{out}}$$

$$\mathbf{h} = f(\mathbf{x}\mathbf{W})$$
$$\mathbf{y} = g(\mathbf{h}\beta)$$

* f, g sind differenzierbar

Beispiel – Anamolieerkennung

Ein Blinker im Computer eines Autos registriert, wenn der Blinkerhebel nach links **L** oder rechts **R** gedrückt wird. Wenn der Hebel nach links gedrückt wird, sollte der Blinker **B** blinken. Dasselbe gilt für den rechten Hebel. Wenn das System jedoch erkennt, dass der Hebel in beide Richtungen gleichzeitig gedrückt wird, liegt wahrscheinlich ein Fehler vor und der Blinker sollte nicht aktiviert werden.

L	**R**	**B**
0	0	0
0	1	1
1	0	1
1	1	0

Dies ist eine XOR-Funktion,
oder in der Logik:

$$(L \vee R) \wedge \neg(L \wedge R) \rightarrow B$$

Beispiel – Anamolieerkennung

Das binäre Klassifizierungsproblem ist **nicht** *linear trennbar*, da für eine perceptron mit die folgenden gewählten Werte $w_1 = 1$, $w_2 = 1$, und $b = -0.5$:

$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b > 0$$

$$\hat{y} = 0 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b < 0$$

L	R	B
0	0	0
0	1	1
1	0	1
1	1	0

$$\hat{y} = 0 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0 \cdot 1 + 0 \cdot 1 + (-0.5) = -0.5 < 0$$

$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0 \cdot 1 + 1 \cdot 1 + (-0.5) = 0.5 > 0$$

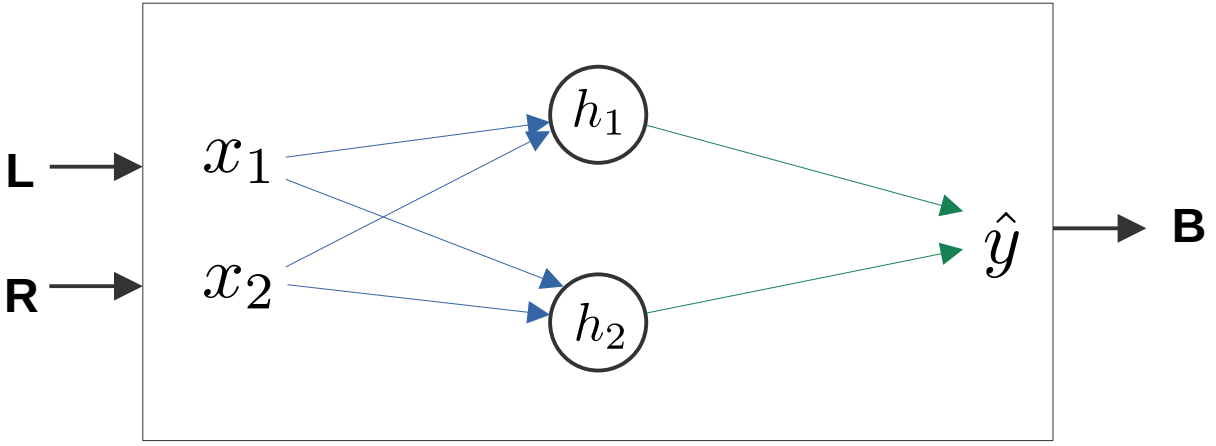
$$\hat{y} = 1 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 1 \cdot 1 + 0 \cdot 1 + (-0.5) = 0.5 > 0$$

$$\hat{y} = 0 \Rightarrow \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 1 \cdot 1 + 1 \cdot 1 + (-0.5) = 1.5$$

Beispiel – Anamolieerkennung

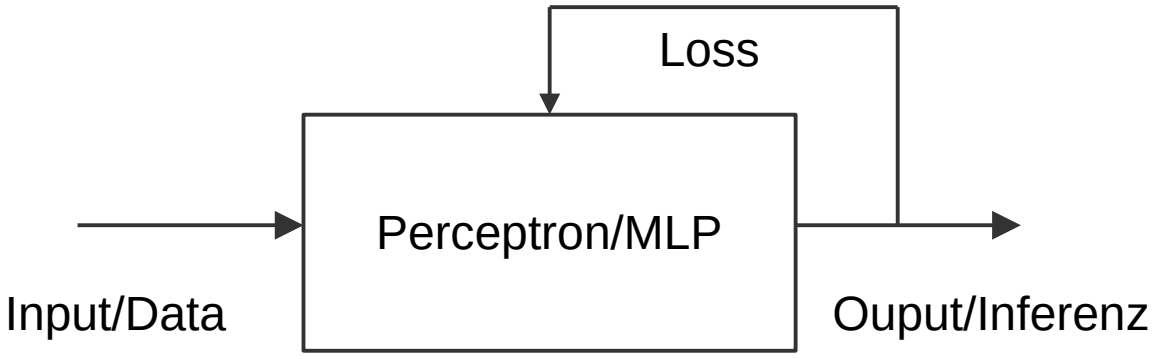
Die folgende MLP könnte die XOR-Funktion implementieren:

L	R	B
0	0	0
0	1	1
1	0	1
1	1	0

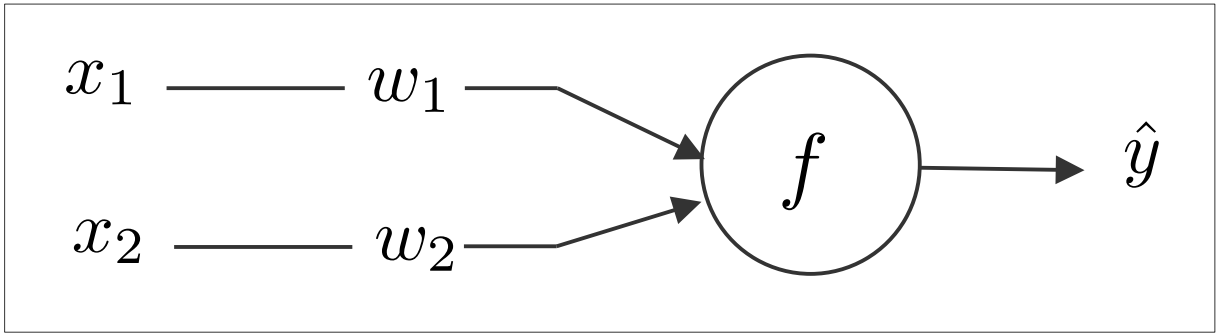


Training und Validation

Bei unserem Modell-Lern-Inferenz-Paradigma können Perceptrons und MLPs als **Modelle** betrachtet werden, die Loss Function wird verwendet, um aus der tatsächlichen Ausgabe und der korrekten Ausgabe aus den markierten Daten zu **lernen**, und die endgültigen trainierten Gewichte ermöglichen es dem Modell, **Inferenz** entlang der approximierten Funktionslinie durchzuführen.



Training Algorithm - Perceptron



1. Loss Function Definieren: $\mathcal{L}(\hat{y}, y) = y - \hat{y}$

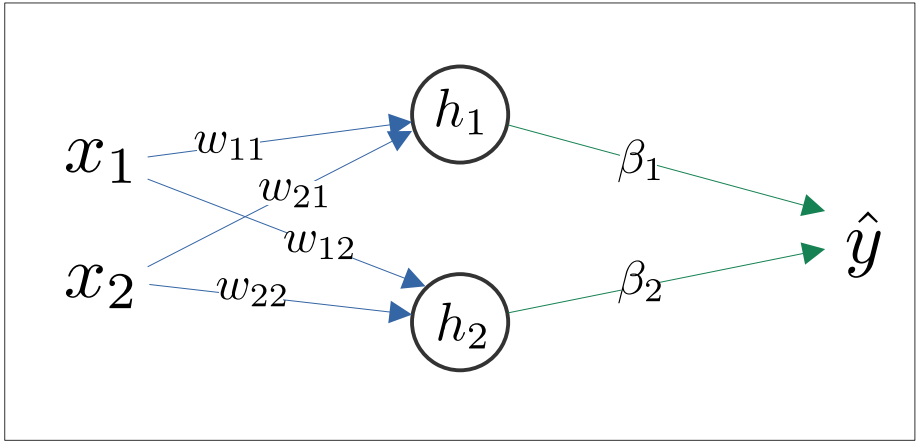
2. Learning rate Definieren: $\eta = 0.1$

3. While (Loss Nicht akzeptabel):

- Berechnung der Ausgabe $\hat{y}$ für die Eingaben x_1, x_2 bzw. der Loss

- Gewichte updaten: $\forall i : w_i = w_i + \eta\mathcal{L}$

Training Algorithm - MLP



In diesem Fall müssen wir für das Training der Gewichte die **Back Propagation** verwenden. Dazu wird ein Forward-Pass durchgeführt, der Gradient der Loss wird berechnet und rückwärts durch das Netz propagiert, um die Gewichte zu aktualisieren.

Training Algorithm - MLP

1. Forward-Pass:

$$h_1 = f(w_{11}x_1 + w_{12}x_2)$$
$$h_2 = f(w_{21}x_1 + w_{22}x_2)$$
$$\hat{y} = g(\beta_1 h_1 + \beta_2 h_2)$$

2. Loss berechnen (MSE) :

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$$

* Frage: Wie können wir die Gewichte so ändern, dass der Verlust reduziert wird?

Training Algorithm - MLP

3. Berechnen Sie den Gradienten in Bezug auf die Output Layer Gewichte. Beispiel: β_1

$$\frac{\partial \mathcal{L}}{\partial \beta_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial (\beta_1 h_1 + \beta_2 h_2)} \frac{\partial (\beta_1 h_1 + \beta_2 h_2)}{\partial \beta_1}$$

- $$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y$$
- $$\frac{\partial \hat{y}}{\partial (\beta_1 h_1 + \beta_2 h_2)} = g'(\beta_1 h_1 + \beta_2 h_2)$$
- $$\frac{\partial (\beta_1 h_1 + \beta_2 h_2)}{\partial \beta_1} = h_1$$

Training Algorithm - MLP

4. Berechnen Sie den Gradienten in Bezug auf die Hidden Layer Gewichte.
Beispiel: w_{11}

- $$\frac{\partial \mathcal{L}}{\partial w_{11}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial (\beta_1 h_1 + \beta_2 h_2)} \frac{\partial (\beta_1 h_1 + \beta_2 h_2)}{\partial h_1} \frac{\partial h_1}{\partial (w_{11} x_1 + w_{12} x_2)} \frac{\partial (w_{11} x_1 + w_{12} x_2)}{\partial w_{11}}$$
- $$\frac{\partial (\beta_1 h_1 + \beta_2 h_2)}{\partial h_1} = \beta_1 \,, \qquad \frac{\partial h_1}{\partial (w_{11} x_1 + w_{12} x_2)} = f'(w_{11} x_1 + w_{12} x_2)$$
- $$\frac{\partial (w_{11} x_1 + w_{12} x_2)}{\partial w_{11}} = x_1$$

Training Algorithm - MLP

5. Aktualisieren Sie jedes Gewicht anhand des zuvor berechneten Gradienten.
Beispiel: w_{11}

$$w_{11} = w_{11} + \eta \frac{\partial \mathcal{L}}{\partial w_{11}}$$

Back Propagation mit H4

Professor H4 hat ein hervorragendes Beispiel für eine handgefertigte Back Prop und ein Python-Beispiel. Sehen Sie es sich hier an:

https://github.com/ProfH4/Lecture_Notes/blob/main/2025%20Sommersemester/AKIB2%20SS25%20Weiterfu%CC%88hrende%20KI%20Programmierung/20250402_AKIB2_Python%20-%2020250402_111416.pdf

https://github.com/ProfH4/Lecture_Notes/blob/main/2025%20Sommersemester/AKIB2%20SS25%20Weiterfu%CC%88hrende%20KI%20Programmierung/20250404_AKIB2_Python_Feed_forward_back_propagation.ipynb