OST
Ostschweizer
Fachhochschule

1 Verlustfunktionen im ML

Systemtechnik BSc
FS 2025

Aufgaben

Applied Neural Networks im Modul ANN (WUCH)

Inhaltsverzeichnis

Abstract

Dieser Artikel bietet einen umfassenden Überblick über Verlustfunktionen als zentrale Komponente der Modelloptimierung im Maschinellen Lernen. Er analysiert zunächst Funktionen für die Regression und beleuchtet deren unterschiedliche Robustheit gegenüber Ausreissern, wie beim Vergleich von MSE und MAE. Anschliessend werden Ansätze für die Klassifikation behandelt, die von Maximum-Margin-Methoden wie dem Hinge-Verlust bis zu probabilistischen Modellen mittels Kreuzentropie reichen. Weiterhin werden kontrastive Verluste für das selbst-überwachte Lernen von Datenrepräsentationen durch den Vergleich ähnlicher und unähnlicher Datenpunkte erläutert. Zuletzt stellt der Artikel adversariale Verluste vor, die das kompetitive Training von Generative Adversarial Networks (GANs) ermöglichen. Der Text verdeutlicht, dass die Wahl der Verlustfunktion eine kritische Designentscheidung ist, die die Leistung, Robustheit und das Verhalten eines Modells massgeblich beeinflusst.

1 Optimierung von Modellen mittels Verlustfunktionen und Gradientenabstieg

Im Rahmen des überwachten maschinellen Lernens ist das primäre Ziel, eine Funktion $f_\theta(\boldsymbol{x})$ zu lernen, die Eingabedaten $\boldsymbol{x}$ möglichst präzise auf zugehörige Zielwerte y abbildet. Die Funktion f_θ wird durch einen Satz von Parametern θ (z.B. die Gewichte und Biases eines neuronalen Netzes) bestimmt. Um zu quantifizieren, wie gut das Modell mit den aktuellen Parametern θ diese Aufgabe erfüllt, wird eine **Verlustfunktion** (Loss Function) $\mathcal{L}$ verwendet.
Die Verlustfunktion $\mathcal{L}(y, \hat{y})$ misst die Diskrepanz oder den "Verlustßwischen dem wahren Zielwert y und der Vorhersage $\hat{y} = f_\theta(\boldsymbol{x})$ für ein einzelnes Datenbeispiel $(\boldsymbol{x}, y)$. Das übergeordnete Ziel des Trainingsprozesses ist es, die Parameter θ des Modells so zu optimieren, dass der durchschnittliche Verlust über den gesamten Trainingsdatensatz $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ minimiert wird. Diese zu minimierende Zielfunktion (Objective Function), oft als $\mathcal{L}(\theta)$ bezeichnet, lautet typischerweise:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i)) \tag{1}$$

Hierbei bezeichnet $\mathcal{L}(\theta)$ den durchschnittlichen Gesamtverlust als Funktion der Parameter θ, während $\mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i))$ den Verlust für das einzelne Beispiel i darstellt. Die Wahl einer geeigneten Verlustfunktion $\mathcal{L}(y, \hat{y})$ hängt maßgeblich von der Art der Lernaufgabe ab. Wie in den folgenden Abschnitten detailliert beschrieben wird, verwendet man für Klassifikationsaufgaben andere Verlustfunktionen (z.B. Kreuzentropie, Hinge-Verlust) als für Regressionsaufgaben (z.B. Mittlerer Quadratischer Fehler, Mittlerer Absoluter Fehler) oder für komplexere Szenarien wie generative Modellierung (z.B. adversariale Verluste) oder das Lernen von Repräsentationen (z.B. kontrastive Verluste). Unabhängig von der spezifischen Wahl der Verlustfunktion benötigen wir ein algorithmisches Verfahren, um die optimalen Parameter θ^* zu finden, die die Zielfunktion $\mathcal{L}(\theta)$ minimieren:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta) \tag{2}$$

Für einfache Modelle wie die lineare Regression existieren analytische Lösungen, aber für komplexe Modelle wie tiefe neuronale Netze ist $\mathcal{L}(\theta)$ typischerweise eine hochdimensionale, nicht-konvexe Funktion, für die analytische Lösungen nicht praktikabel

sind. Hier kommen iterative Optimierungsverfahren ins Spiel.

1.1 Das Gradientenabstiegsverfahren (Gradient Descent)

Das bei weitem am häufigsten verwendete Optimierungsverfahren im maschinellen Lernen, insbesondere im Deep Learning, ist das **Gradientenabstiegsverfahren** (Gradient Descent). Die Grundidee ist einfach: Man startet mit einer anfänglichen Schätzung der Parameter θ_0 und bewegt sich dann iterativ in kleinen Schritten in die Richtung, die den Wert der Verlustfunktion $\mathcal{L}(\theta)$ am stärksten reduziert.
Herleitung: Wir möchten die Parameter θ so ändern, dass der Wert der Zielfunktion $\mathcal{L}(\theta)$ sinkt. Angenommen, wir befinden uns beim Parametervektor θ_k im k-ten Iterationsschritt. Wir suchen eine kleine Änderung $\Delta\theta$, sodass $\mathcal{L}(\theta_k + \Delta\theta) < \mathcal{L}(\theta_k)$ gilt.
Mittels einer Taylor-Entwicklung erster Ordnung können wir $\mathcal{L}(\theta_k + \Delta\theta)$ in der Nähe von θ_k approximieren:

$$\mathcal{L}(\theta_k + \Delta\theta) \approx \mathcal{L}(\theta_k) + \nabla_\theta \mathcal{L}(\theta_k)^T \Delta\theta \tag{3}$$

Hierbei ist $\nabla_\theta \mathcal{L}(\theta_k)$ der Gradientenvektor der Zielfunktion $\mathcal{L}$ bezüglich der Parameter θ, ausgewertet an der Stelle θ_k. Der Gradient $\nabla_\theta \mathcal{L}(\theta_k)$ zeigt in die Richtung des steilsten Anstiegs der Funktion $\mathcal{L}$ an der Stelle θ_k.
Damit $\mathcal{L}(\theta_k + \Delta\theta) < \mathcal{L}(\theta_k)$ gilt, muss der zweite Term in Gl. (3) negativ sein:

$$\nabla_\theta \mathcal{L}(\theta_k)^T \Delta\theta < 0 \tag{4}$$

Um den Wert von $\mathcal{L}$ möglichst schnell zu reduzieren, suchen wir die Richtung $\Delta\theta$, die bei einer festen (kleinen) Schrittlänge $\|\Delta\theta\|$ den Wert des Skalarprodukts $\nabla_\theta \mathcal{L}(\theta_k)^T \Delta\theta$ minimiert. Das Skalarprodukt $\boldsymbol{a}^T\boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\| \cos\phi$ wird minimal (am negativsten), wenn der Winkel ϕ zwischen den Vektoren $\nabla_\theta \mathcal{L}(\theta_k)$ und $\Delta\theta$ gleich $180°$ ist, d.h., wenn $\Delta\theta$ in die genau entgegengesetzte Richtung des Gradienten zeigt.
Wir wählen daher die Aktualisierungsrichtung als den negativen Gradienten:

$$\Delta\theta = -\eta \nabla_\theta \mathcal{L}(\theta_k) \tag{5}$$

Hier ist $\eta > 0$ ein kleiner positiver Skalar, der als **Lernrate** (Learning Rate) bezeichnet wird. Die Lernrate steuert die Schrittweite bei jedem Aktualisierungsschritt.

Eine zu große Lernrate kann dazu führen, dass der Algorithmus über das Minimum hinausschießt und divergiert, während eine zu kleine Lernrate die Konvergenz stark verlangsamt.

Die Update-Regel: Kombiniert man die aktuelle Parameterschätzung θ_k mit der Änderung $\Delta\theta$, ergibt sich die iterative Update-Regel des Gradientenabstiegs:

$$\theta_{k+1} = \theta_k + \Delta\theta = \theta_k - \eta \nabla_\theta \mathcal{L}(\theta_k) \tag{6}$$

Dieser Schritt wird wiederholt, bis ein Konvergenzkriterium erfüllt ist, z.B. wenn der Gradient sehr klein wird ($\|\nabla_\theta \mathcal{L}(\theta_k)\| \approx 0$), die Änderung der Parameter oder des Verlusts unter einen Schwellenwert fällt, oder eine maximale Anzahl von Iterationen erreicht ist.

Algorithmus (Allgemeine Form):

(a) Initialisiere die Parameter θ_0 (z.B. zufällig).

(b) Wiederhole für $k = 0, 1, 2, \ldots$ bis Konvergenz:

   a) Berechne den Gradienten der Zielfunktion: $\boldsymbol{g}_k = \nabla_\theta \mathcal{L}(\theta_k)$.

   b) Aktualisiere die Parameter: $\theta_{k+1} = \theta_k - \eta \boldsymbol{g}_k$.

(c) Gebe die optimierten Parameter θ_{k+1} zurück.

1.2 Varianten des Gradientenabstiegs

Die Berechnung des exakten Gradienten $\nabla_\theta \mathcal{L}(\theta)$ erfordert gemäß Gl. (1) die Berechnung des Verlusts und seines Gradienten für *jedes einzelne* Beispiel im Trainingsdatensatz D:

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \left(\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i)) \right) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i)) \tag{7}$$

Für sehr große Datensätze (z.B. Millionen von Bildern) ist die Berechnung dieses vollständigen Gradienten in jedem Iterationsschritt extrem rechenaufwändig und möglicherweise unpraktikabel. Aus diesem Grund wurden verschiedene Varianten des Gradientenabstiegs entwickelt.

1.2.1 Batch Gradient Descent (BGD)

Erklärung: Dies ist die Standardvariante, die oben beschrieben wurde. Der Gradient wird über den *gesamten* Trainingsdatensatz berechnet, bevor ein einziger Parameterschritt durchgeführt wird.
Formel (Update):

$$\theta \leftarrow \theta - \eta \left(\frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i)) \right) \tag{8}$$

Vorteile:

- Der Gradient ist eine exakte Schätzung des wahren Gradienten der Zielfunktion $\mathcal{L}(\theta)$.
- Die Konvergenz ist oft stabil und direkt auf ein lokales (bei konvexen Problemen globales) Minimum gerichtet.

Nachteile:

- Sehr langsam und rechenintensiv für große Datensätze, da alle Daten für jeden Schritt verarbeitet werden müssen.
- Möglicherweise nicht durchführbar, wenn der Datensatz nicht in den Speicher passt.
- Kann in flachen lokalen Minima stecken bleiben.

1.2.2 Stochastic Gradient Descent (SGD)

Erklärung: Beim Stochastischen Gradientenabstieg wird der Gradient für die Parameteraktualisierung basierend auf *nur einem einzigen*, zufällig ausgewählten Trainingsbeispiel $(\boldsymbol{x}_i, y_i)$ in jedem Schritt geschätzt.
Formel (Update für Beispiel i):

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i)) \tag{9}$$

Innerhalb einer Trainingsepoche (ein Durchlauf durch den gesamten Datensatz) werden also N Parameter-Updates durchgeführt.
Vorteile:

- Deutlich schnellere Updates (N Updates pro Epoche vs. 1 bei BGD).
- Geringer Speicherbedarf pro Update (nur ein Beispiel).
- Die hohe Varianz ("Rauschen") im Gradienten kann helfen, aus flachen lokalen Minima zu entkommen und potenziell bessere Minima zu finden.
- Ermöglicht Online-Lernen (Modellaktualisierung bei Eintreffen neuer Daten).

Nachteile:

- Hohe Varianz der Gradientenschätzung führt zu stark oszillierendem Konvergenzpfad.
- Konvergiert nicht exakt zum Minimum, sondern oszilliert typischerweise darum herum (es sei denn, die Lernrate wird über die Zeit reduziert).
- Verliert Effizienzvorteile durch vektorisierte Operationen auf moderner Hardware (GPUs).

1.2.3 Mini-Batch Gradient Descent (MBGD)

Erklärung: Dies ist der am häufigsten verwendete Kompromiss zwischen BGD und SGD. Der Gradient wird über einen kleinen, zufällig ausgewählten Teildatensatz, den sogenannten **Mini-Batch** $\mathcal{B}$ der Größe B (wobei $1 < B < N$), berechnet. Typische Batch-Größen liegen im Bereich von $B = 32$ bis $B = 512$.
Formel (Update für Mini-Batch $\mathcal{B}$):

$$\theta \leftarrow \theta - \eta \left(\frac{1}{B} \sum_{i \in \mathcal{B}} \nabla_\theta \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i)) \right) \tag{10}$$

Eine Epoche besteht aus $\lceil N/B \rceil$ Updates.
Vorteile:

- Reduziert die Varianz der Gradientenschätzung im Vergleich zu SGD, was zu stabilerer Konvergenz führt.
- Nutzt die Vorteile der Vektorisierung und parallelen Verarbeitung auf GPUs effizient aus.

- Schneller als BGD und oft stabiler/effizienter als reines SGD.

Nachteile:

- Einführung eines neuen Hyperparameters (Batch-Größe B), der abgestimmt werden muss.
- Der Gradient ist immer noch eine Schätzung (weniger verrauscht als SGD, aber nicht exakt wie BGD).

Die hier vorgestellten Varianten des Gradientenabstiegs bilden die Grundlage für die Optimierung der meisten modernen Modelle des maschinellen Lernens. Aufbauend darauf wurden zahlreiche Weiterentwicklungen vorgeschlagen, um die Konvergenzgeschwindigkeit und -stabilität weiter zu verbessern. Dazu gehören Techniken wie die Verwendung von **Momentum** (um Oszillationen zu dämpfen und die Konvergenz zu beschleunigen) oder **adaptive Lernratenverfahren** (wie AdaGrad, RMSprop und Adam), die die Lernrate η für jeden Parameter individuell anpassen.
Die spezifische Berechnung des Gradienten $\nabla_\theta \mathcal{L}(y_i, f_\theta(\boldsymbol{x}_i))$ hängt natürlich von der gewählten Verlustfunktion $\mathcal{L}$ und der Architektur des Modells f_θ ab. Für neuronale Netze wird dieser Gradient effizient mittels des **Backpropagation**-Algorithmus berechnet, welcher im Wesentlichen die Kettenregel der Differentialrechnung anwendet. Die Details der spezifischen Verlustfunktionen für verschiedene Aufgaben werden in den folgenden Abschnitten behandelt.

2 Verlustfunktionen für die Regression

Bei Regressionsproblemen im überwachten Lernen ist das Ziel, eine kontinuierliche Zielvariable $y \in \mathbb{R}$ basierend auf Eingabemerkmalen $\boldsymbol{x}$ vorherzusagen. Ein Regressionsmodell f_θ lernt eine Funktion, die eine Vorhersage $\hat{y} = f_\theta(\boldsymbol{x})$ für einen gegebenen Input $\boldsymbol{x}$ liefert.
Die **Verlustfunktion** spielt hierbei die entscheidende Rolle, die Diskrepanz oder den Fehler zwischen dem wahren Wert y und dem vorhergesagten Wert $\hat{y}$ zu quantifizieren. Das Ziel des Trainings ist es, die Parameter θ des Modells so anzupassen, dass der durchschnittliche Verlust über den Trainingsdatensatz minimiert wird.

Die Wahl der Verlustfunktion beeinflusst nicht nur die Konvergenz des Trainingsprozesses, sondern auch die Eigenschaften der resultierenden Vorhersagen (z.B. ob das Modell tendenziell den Mittelwert oder den Median vorhersagt) und die Robustheit des Modells gegenüber Ausreissern in den Daten.
Wir verwenden die folgende Notation: y_i ist der wahre Wert für das i-te Beispiel, $\hat{y}_i$ ist der vom Modell vorhergesagte Wert, und N ist die Anzahl der Beispiele im Datensatz. Der Fehler oder das Residuum für ein Beispiel ist $\varepsilon_i = y_i - \hat{y}_i$.

2.1 Mittlerer Quadratischer Fehler (Mean Squared Error, MSE / L2-Verlust)

Der Mittlere Quadratische Fehler (MSE), auch L2-Verlust genannt, ist die am häufigsten verwendete Verlustfunktion für Regressionsprobleme.
Erklärung: MSE berechnet den Durchschnitt der quadrierten Differenzen zwischen den wahren und den vorhergesagten Werten. Durch das Quadrieren werden grössere Fehler überproportional stark bestraft. Dies macht MSE sehr empfindlich gegenüber Ausreissern.
Formel:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2 \tag{11}$$

Für einen einzelnen Datenpunkt wird der Verlust oft als $(y - \hat{y})^2 = \varepsilon^2$ betrachtet.
Herleitung/Motivation: MSE ergibt sich natürlich aus der Maximum-Likelihood-Schätzung (MLE), wenn angenommen wird, dass die Fehler $\varepsilon_i = y_i - \hat{y}_i$ unabhängig und identisch normalverteilt (Gaussverteilung) mit Mittelwert Null und konstanter Varianz sind. Unter dieser Annahme maximiert die Minimierung des MSE die Plausibilität der Modellparameter gegeben die Daten. Mathematisch ist MSE attraktiv, da die Funktion konvex und glatt (unendlich oft differenzierbar) ist, was die Optimierung mit gradientenbasierten Methoden erleichtert. Die Ableitung nach $\hat{y}_i$ ist einfach $-2(y_i - \hat{y}_i) = 2(\hat{y}_i - y_i) = -2\varepsilon_i$. Modelle, die mit MSE trainiert werden, lernen tendenziell, den bedingten *Mittelwert* von y gegeben $\boldsymbol{x}$ vorherzusagen.
Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Minimiere den Durchschnitt der quadrierten Fehler.
Formel (pro Punkt)	$(y-\hat{y})^2 = \varepsilon^2$.
Ableitung (nach $\hat{y}$)	$2(\hat{y}-y) = -2\varepsilon$.
Vorteile	▪ Mathematisch einfach zu handhaben (konvex, glatt, einfache Ableitung). ▪ Starke Verbindung zur MLE bei Gaussschem Rauschen. ▪ Optimale Vorhersage ist der bedingte Mittelwert.
Nachteile/ Herausforderungen	▪ Sehr empfindlich gegenüber Ausreissern aufgrund der Quadrierung grosser Fehler. ▪ Kann zu verzerrten Modellen führen, wenn Ausreisser vorhanden sind.
Use Cases	Standard-Verlustfunktion für viele Regressionsalgorithmen (z.B. Lineare Regression, Neuronale Netze), wenn keine starken Ausreisser erwartet werden oder der Mittelwert von Interesse ist.

2.2 Mittlerer Absoluter Fehler (Mean Absolute Error, MAE / L1-Verlust)

Der Mittlere Absolute Fehler (MAE), auch L1-Verlust genannt, ist eine Alternative zu MSE, die robuster gegenüber Ausreissern ist.

Erklärung: MAE berechnet den Durchschnitt der absoluten Differenzen zwischen den wahren und den vorhergesagten Werten. Da die Fehler linear und nicht quadratisch gewichtet werden, haben Ausreisser einen geringeren Einfluss auf den Gesamtverlust als bei MSE.

Formel:

$$\mathcal{L}_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| = \frac{1}{N}\sum_{i=1}^{N}|\varepsilon_i| \tag{12}$$

Für einen einzelnen Datenpunkt ist der Verlust $|y-\hat{y}| = |\varepsilon|$.

Herleitung/Motivation: MAE entspricht der MLE, wenn angenommen wird, dass die Fehler einer Laplace-Verteilung folgen. Ein Modell, das trainiert wird, um MAE zu minimieren, lernt, den bedingten *Median* von y gegeben $\boldsymbol{x}$ vorherzusagen. Der Median ist bekanntermassen robuster gegenüber Ausreissern als der Mittelwert. Ein Nachteil ist, dass die MAE-Funktion am Punkt $y = \hat{y}$ (Fehler $\varepsilon = 0$) nicht differenzierbar ist (die Ableitung springt von -1 auf $+1$). In der Praxis verwendet man Subgradienten (z.B. 0 oder ± 1) oder glättet die Funktion nahe null. Die Ableitung nach $\hat{y}_i$ ist $-\text{sgn}(y_i - \hat{y}_i) = -\text{sgn}(\varepsilon_i)$.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Minimiere den Durchschnitt der absoluten Fehler.
Formel (pro Punkt)	$\lvert y - \hat{y} \rvert = \lvert \varepsilon \rvert$.
Ableitung (nach $\hat{y}$)	$\text{sgn}(\hat{y} - y) = -\text{sgn}(\varepsilon)$ (definiert als 0 oder ± 1 bei $\varepsilon = 0$).
Vorteile	▪ Deutlich robuster gegenüber Ausreissern als MSE. ▪ Optimale Vorhersage ist der bedingte Median. ▪ Intuitive Interpretation (durchschnittlicher absoluter Fehler).
Nachteile/ Herausforderungen	▪ Nicht differenzierbar bei Null-Fehler (erfordert Subgradienten oder Glättung). ▪ Kann zu langsamerer Konvergenz führen, da der Gradient konstant (± 1) ist und nicht kleiner wird, wenn man sich dem Minimum nähert.
Use Cases	Regression bei Vorhandensein von Ausreissern, Vorhersage des Medians, Situationen, in denen grosse Fehler nicht überproportional bestraft werden sollen.

2.3 Huber-Verlust

Der Huber-Verlust ist eine hybride Verlustfunktion, die versucht, die Vorteile von MSE und MAE zu kombinieren.

Erklärung: Der Huber-Verlust verhält sich wie MSE für kleine Fehler (innerhalb eines Schwellenwerts $\pm\delta$) und wie MAE (linear) für grosse Fehler. Der Parameter δ steuert den Übergangspunkt. Dadurch ist der Huber-Verlust weniger empfindlich gegenüber Ausreissern als MSE, aber immer noch differenzierbar am Nullpunkt (im Gegensatz zu MAE).

Formel: Für einen einzelnen Fehler $\varepsilon = y - \hat{y}$:

$$\mathcal{L}_{\text{Huber}}(\varepsilon, \delta) = \begin{cases} \frac{1}{2}\varepsilon^2 & \text{für } |\varepsilon| \leq \delta \\ \delta(|\varepsilon| - \frac{1}{2}\delta) & \text{für } |\varepsilon| > \delta \end{cases} \tag{13}$$

Der Gesamtverlust ist der Durchschnitt über alle Datenpunkte.

Motivation: Ziel ist es, die Robustheit von MAE für grosse Fehler mit der Effizienz und Glattheit von MSE für kleine Fehler zu verbinden. Die Funktion ist stetig differenzierbar.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Quadratischer Verlust für kleine Fehler, linearer Verlust für grosse Fehler.
Formel (pro Punkt, $\varepsilon = y - \hat{y}$)	Siehe Gl. (13).
Ableitung (nach $\hat{y}$, für $\varepsilon = y - \hat{y}$)	$\begin{cases} -\varepsilon & \text{für } \lvert\varepsilon\rvert \leq \delta \\ -\delta \cdot \text{sgn}(\varepsilon) & \text{für } \lvert\varepsilon\rvert > \delta \end{cases}$.
Parameter	$\delta > 0$ (Schwellenwert für den Übergang).
Vorteile	▪ Guter Kompromiss zwischen MSE und MAE. ▪ Weniger empfindlich gegenüber Ausreissern als MSE. ▪ Stetig differenzierbar (im Gegensatz zu MAE).
Nachteile/ Herausforderungen	▪ Erfordert die Wahl des Hyperparameters δ. ▪ Komplexere Formel als MSE oder MAE.
Use Cases	Robuste Regression, wenn Ausreisser erwartet werden, aber die Glattheit von MSE wünschenswert ist. Oft in Verstärkungslernen (Reinforcement Learning) verwendet.

2.4 Log-Cosh-Verlust

Der Log-Cosh-Verlust ist eine weitere glatte Verlustfunktion, die sich ähnlich wie MAE verhält, aber überall zweimal differenzierbar ist.

Erklärung: Er basiert auf dem Logarithmus des hyperbolischen Kosinus des Fehlers. Für kleine Fehler ε approximiert $\log(\cosh(\varepsilon))$ den quadratischen Fehler $\frac{1}{2}\varepsilon^2$, während es für grosse Fehler dem absoluten Fehler $|\varepsilon| - \log 2$ ähnelt.

Formel:

$$\mathcal{L}_{\text{LogCosh}} = \frac{1}{N}\sum_{i=1}^{N} \log(\cosh(\hat{y}_i - y_i)) = \frac{1}{N}\sum_{i=1}^{N} \log(\cosh(\varepsilon_i)) \tag{14}$$

(Beachte: $\cosh(-x) = \cosh(x)$)

Motivation: Ziel ist es, eine Verlustfunktion zu haben, die die Robustheitseigenschaften von MAE/Huber besitzt, aber sehr glatt ist (unendlich oft differenzierbar), was für manche Optimierungsalgorithmen (z.B. solche, die zweite Ableitungen nutzen) vorteilhaft sein kann.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Glatte (zweimal differenzierbare) Annäherung an MAE.
Formel (pro Punkt)	$\log(\cosh(\hat{y} - y)) = \log(\cosh(\varepsilon))$.
Ableitung (nach $\hat{y}$)	$\tanh(\hat{y} - y) = -\tanh(\varepsilon)$.
Vorteile	▪ Glatt (unendlich oft differenzierbar). ▪ Robustheit ähnlich zu Huber/MAE. ▪ Keine zusätzlichen Hyperparameter wie δ.
Nachteile/ Herausforderungen	▪ Weniger gebräuchlich oder intuitiv als MSE/MAE/Huber. ▪ Berechnung von $\cosh$ und $\tanh$ kann numerisch aufwendiger sein.
Use Cases	Robuste Regression, wenn Glattheit (zweite Ableitung) wichtig ist. Alternative zu Huber, wenn keine Parameterabstimmung gewünscht ist.

2.5 Quantil-Verlust (Pinball Loss)

Der Quantil-Verlust, auch Pinball Loss genannt, wird verwendet, um bedingte Quantile (anstelle des Mittelwerts oder Medians) der Zielvariablen vorherzusagen.
Erklärung: Quantilregression ermöglicht es, verschiedene Punkte der bedingten Verteilung von y zu modellieren, z.B. das 10., 50. (Median) oder 90. Perzentil. Der Quantil-Verlust ist asymmetrisch und bestraft Über- und Unterschätzungen unterschiedlich, abhängig vom Zielquantil $\tau \in (0, 1)$.
Formel: Für einen Fehler $\varepsilon = y - \hat{y}$ und ein Zielquantil τ:

$$\mathcal{L}_{\text{Quantile}}(\varepsilon, \tau) = \begin{cases} \tau\varepsilon & \text{für } \varepsilon \geq 0 \quad (\text{Unterschätzung } \hat{y} < y) \\ (\tau - 1)\varepsilon & \text{für } \varepsilon < 0 \quad (\text{Überschätzung } \hat{y} > y) \end{cases} \tag{15}$$

Dies kann auch kompakt als $\max(\tau\varepsilon, (\tau-1)\varepsilon)$ geschrieben werden. Der Gesamtverlust ist der Durchschnitt über alle Datenpunkte.
Motivation: Für $\tau = 0.5$ ist der Verlust $\max(0.5\varepsilon, -0.5\varepsilon) = 0.5|\varepsilon|$, was äquivalent zu MAE ist (Minimierung führt zur Vorhersage des Medians). Für $\tau > 0.5$ werden Unterschätzungen ($\varepsilon > 0$) stärker bestraft als Überschätzungen ($\varepsilon < 0$), was das Modell dazu bringt, höhere Quantile vorherzusagen. Für $\tau < 0.5$ ist es umgekehrt. Dies ist nützlich, um Unsicherheitsintervalle zu schätzen oder Risiken zu modellieren.
Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Asymmetrische Bestrafung von Fehlern zur Vorhersage spezifischer bedingter Quantile.
Formel (pro Punkt, $\varepsilon = y - \hat{y}$)	$\max(\tau\varepsilon, (\tau - 1)\varepsilon)$.
Parameter	$\tau \in (0, 1)$ (Zielquantil).
Vorteile	▪ Ermöglicht Vorhersage beliebiger Quantile, nicht nur des Mittelwerts/Medians. ▪ Gibt Einblick in die bedingte Verteilung und Unsicherheit. ▪ Robust gegenüber Ausreissern (wie MAE).
Nachteile/ Herausforderungen	▪ Nicht differenzierbar bei Null-Fehler ($\varepsilon = 0$, wie MAE). ▪ Erfordert die Wahl des Quantils τ. ▪ Vorhersage einzelner Quantile kann weniger stabil sein als Mittelwert-/Medianvorhersage.
Use Cases	Quantilregression, Schätzung von Vorhersageintervallen, Risikomodellierung (z.B. Value-at-Risk), Ökonometrie, überall dort, wo die gesamte Verteilung von Interesse ist.

2.6 Vergleich von Regressions-Verlustfunktionen

Die Wahl der Verlustfunktion in der Regression ist ein wichtiger Aspekt des Modelldesigns. MSE ist der Standard aufgrund seiner mathematischen Einfachheit und Verbindung zur Gauss-Annahme, aber seine Empfindlichkeit gegenüber Ausreissern ist ein signifikanter Nachteil in vielen realen Anwendungen. MAE bietet Robustheit, opfert aber die Differenzierbarkeit am Minimum. Huber und Log-Cosh stellen Kompromisse dar, die Robustheit mit Glattheit verbinden, wobei Huber einen expliziten Parameter δ benötigt. Der Quantil-Verlust erweitert den Fokus von zentralen Tendenzmassen (Mittelwert, Median) auf die gesamte bedingte Verteilung und ist unerlässlich für Aufgaben wie die Schätzung von Unsicherheitsintervallen. Die Entscheidung sollte basierend auf den Eigenschaften der Daten (insbesondere dem Vorhandensein von Ausreissern) und dem spezifischen Ziel der Regressionsanalyse getroffen werden (z.B. Vorhersage des Durchschnitts, des wahrscheinlichsten Werts oder eines bestimmten Quantils).
Tabelle 1 fasst die Hauptmerkmale zusammen.

3 Verlustfunktionen für die Klassifikation

Beim überwachten Lernen für die Klassifikation ist das Ziel, eine Abbildung $f : \mathcal{X} \to \mathcal{Y}$ von einem Eingaberaum $\mathcal{X}$ (z.B. $\mathbb{R}^d$) in einen diskreten Ausgaberaum $\mathcal{Y}$, der die Klas-

Tabelle 1: Vergleich von Regressions-Verlustfunktionen

Verlustfunktion	Formel (pro Punkt, $\varepsilon = y - \hat{y}$)	Optimalvorhersage	Robustheit ggü. Ausreissern	Differenzierbarkeit	Hauptanwendung
MSE (L2)	ε^2	Bedingter Mittelwert	Gering	Ja (glatt)	Standardregression, Gauss-Rauschen
MAE (L1)	$\lvert\varepsilon\rvert$	Bedingter Median	Hoch	Nein (bei $\varepsilon = 0$)	Robuste Regression, Medianvorhersage
Huber	$\begin{cases} \frac{1}{2}\varepsilon^2 & \lvert\varepsilon\rvert \le \delta \\ \delta(\lvert\varepsilon\rvert - \frac{1}{2}\delta) & \lvert\varepsilon\rvert > \delta \end{cases}$	Kompromiss Mittelwert/Median	Mittel-Hoch	Ja (stetig diff.)	Robuste Regression (Kompromiss)
Log-Cosh	$\log(\cosh(\varepsilon))$	Ähnlich Median	Mittel-Hoch	Ja (glatt)	Robuste Regression (glatt)
Quantil (Pinball)	$\max(\tau\varepsilon, (\tau - 1)\varepsilon)$	Bedingtes τ-Quantil	Hoch	Nein (bei $\varepsilon = 0$)	Quantilregression, Unsicherheitsschätzung

Hinweis: $\varepsilon = y - \hat{y}$. Robustheit ist relativ. Differenzierbarkeit bezieht sich auf die Stetigkeit der ersten Ableitung.

senlabels repräsentiert, zu lernen. Für eine gegebene Eingabe $\boldsymbol{x}$ erzeugt das Modell f eine Vorhersage, die ein Rohwert (Score) $f(\boldsymbol{x}) \in \mathbb{R}$, eine Wahrscheinlichkeitsverteilung $\hat{\boldsymbol{p}} \in [0,1]^K$ oder ein direktes Klassenlabel $\hat{y} \in \mathcal{Y}$ sein kann. Eine **Verlustfunktion**, $\mathcal{L}(y, \hat{y})$ oder $\mathcal{L}(y, f(\boldsymbol{x}))$, quantifiziert die Kosten (den „Verlust"), die entstehen, wenn das wahre Label y ist und die Vorhersage $\hat{y}$ bzw. aus $f(\boldsymbol{x})$ abgeleitet ist. Das Ziel während des Trainings ist typischerweise die Minimierung des durchschnittlichen Verlusts über den Trainingsdatensatz.

Obwohl das ultimative Ziel bei der Klassifikation oft die Minimierung der Anzahl von Fehlklassifikationen ist (gemessen durch den **Null-Eins-Verlust**), ist diese Verlustfunktion nicht konvex und lässt sich nur schwer direkt mit gradientenbasierten Methoden optimieren. Daher werden verschiedene **Surrogat-Verlustfunktionen** (auch Ersatz-Verlustfunktionen genannt) verwendet, die typischerweise konvex und differenzierbar sind und als Annäherungen an den Null-Eins-Verlust dienen.

Wir betrachten hauptsächlich zwei gängige Konventionen für Labels:

(a) **Binäre Klassifikation mit** $y \in \{-1, +1\}$: Hier gibt das Modell oft einen reellwertigen Score $f(\boldsymbol{x})$ aus. Das Vorzeichen von $f(\boldsymbol{x})$ bestimmt typischerweise die vorhergesagte Klasse $\hat{y} = \text{sgn}(f(\boldsymbol{x}))$, und der Betrag $|f(\boldsymbol{x})|$ kann als Konfidenz interpretiert werden.

(b) **Binäre/Multiklassen-Klassifikation mit Wahrscheinlichkeiten**: Hier wird das wahre Label y oft als ganze Zahl $y \in \{0, 1, \ldots, K-1\}$ oder als One-Hot-Vektor $\boldsymbol{y} \in \{0,1\}^K$ dargestellt. Das Modell gibt eine Wahrscheinlichkeit (für binär, $\hat{p} = P(Y = 1|\boldsymbol{x})$) oder eine Wahrscheinlichkeitsverteilung $\hat{\boldsymbol{p}} = (\hat{p}_0, \ldots, \hat{p}_{K-1})$ aus, wobei $\hat{p}_k = P(Y = k|\boldsymbol{x})$. Der Score $f(\boldsymbol{x})$ oder Vektor

$\boldsymbol{z} = (z_0, \ldots, z_{K-1})$ repräsentiert oft die Werte vor der Aktivierungsfunktion (Logits), bevor eine Funktion wie Sigmoid oder Softmax angewendet wird.

Im Folgenden untersuchen wir die wichtigsten Verlustfunktionen für die Klassifikation.

3.1 Null-Eins-Verlust (Zero-One Loss)

Der Null-Eins-Verlust misst direkt den Klassifikationsfehler. Er weist einer Fehlklassifikation einen Verlust von 1 und einer korrekten Klassifikation einen Verlust von 0 zu.

Formel: Mit vorhergesagtem Label $\hat{y}$:

$$\mathcal{L}_{0-1}(y, \hat{y}) = \mathbb{I}\left[y \neq \hat{y}\right] \tag{16}$$

Mit Score $f(\boldsymbol{x})$ für $y \in \{-1, +1\}$ (unter Annahme der Vorhersage $\hat{y} = \text{sgn}(f(\boldsymbol{x}))$):

$$\mathcal{L}_{0-1}(y, f(\boldsymbol{x})) = \mathbb{I}\left[y \cdot f(\boldsymbol{x}) \leq 0\right] \tag{17}$$

Hier ist $\mathbb{I}\left[\cdot\right]$ die Indikatorfunktion, die 1 zurückgibt, wenn die Bedingung wahr ist, und 0 sonst. Der Term $y \cdot f(\boldsymbol{x})$ ist genau dann positiv, wenn die Vorhersage das korrekte Vorzeichen hat.

Herleitung: Diese Verlustfunktion ist definitorisch und spiegelt direkt das Ziel der Minimierung von Fehlklassifikationen wider.

Eigenschaften und Anwendungsfälle:

Eigenschaft	Beschreibung
Vorteile	▪ Entspricht direkt der Klassifikationsgenauigkeit (Accuracy = $1 -$ Durchschnittlicher $\mathcal{L}_{0-1}$). ▪ Einfache Interpretation.
Nachteile	▪ Nicht konvex. ▪ Nicht differenzierbar (oder Gradient ist fast überall null), was sie für gradientenbasierte Optimierung ungeeignet macht.
Use Cases	Hauptsächlich zur Evaluierung der finalen Modellleistung, nicht zur direkten Optimierung während des Trainings. Andere Verlustfunktionen dienen als Surrogat.

3.2 Hinge-Verlust (Hinge Loss)

Der Hinge-Verlust wird hauptsächlich in Verbindung mit Support-Vektor-Maschinen (SVMs) und der Maximum-Margin-Klassifikation verwendet. Er bestraft Vorhersagen, die falsch sind oder korrekt sind, aber innerhalb der Marge liegen.

Formel: (für $y \in \{-1, +1\}$ und Score $f(\boldsymbol{x})$)

$$\mathcal{L}_{\text{Hinge}}(y, f(\boldsymbol{x})) = \max(0, 1 - y \cdot f(\boldsymbol{x})) \tag{18}$$

Der Term $m = y \cdot f(\boldsymbol{x})$ wird oft als Margin-Score bezeichnet. Der Verlust ist null, wenn der Punkt korrekt mit einer Marge von mindestens 1 klassifiziert wird ($m \geq 1$). Andernfalls steigt der Verlust linear mit dem negativen Margin-Score.
Herleitung: Der Hinge-Verlust ergibt sich aus der Formulierung von Soft-Margin-SVMs. Ziel ist es, eine Hyperebene $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ zu finden, sodass $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i$ für Schlupfvariablen $\xi_i \geq 0$ gilt. Die Minimierung einer Kombination aus der Margengrösse ($\|\boldsymbol{w}\|^2$) und der Gesamtsumme der Schlupfvariablen $\sum \xi_i$ führt zur Minimierung von $\|\boldsymbol{w}\|^2 + C \sum \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b))$, wobei $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$ und der zweite Term den Hinge-Verlust verwendet.
Eigenschaften und Anwendungsfälle:

Eigenschaft	Beschreibung
Vorteile	▪ Konvexe obere Schranke des Null-Eins-Verlusts. ▪ Fördert korrekte Klassifikation mit einer Marge, was potenziell zu besserer Generalisierung führt (Maximum-Margin-Prinzip). ▪ Weniger empfindlich gegenüber Ausreissern als quadratische Verlustfunktionen. ▪ Führt zu dünn besetzten Lösungen bei SVMs (nur Stützvektoren tragen direkt bei).
Nachteile	▪ Nicht differenzierbar bei $y \cdot f(\boldsymbol{x}) = 1$ (Subgradientenverfahren werden zur Optimierung verwendet). ▪ Liefert keine gut kalibrierten Wahrscheinlichkeitsschätzungen. Die Ausgabe $f(\boldsymbol{x})$ ist nur ein Score.
Use Cases	Standard-Verlustfunktion für das Training von linearen SVMs und Kernel-SVMs.

3.3 Logistischer Verlust (Binäre Kreuzentropie)

Der Logistische Verlust, auch bekannt als Log-Verlust oder Binäre Kreuzentropie (Binary Cross-Entropy), wird häufig in der Logistischen Regression und in neuronalen Netzen für die binäre Klassifikation verwendet. Er leitet sich vom Prinzip der Maximum-Likelihood-Schätzung unter Annahme einer Bernoulli-Verteilung für die Labels ab.
Formel: Es gibt zwei gebräuchliche Formen, abhängig von der Label- und Ausgabedarstellung.

1. Labels $y \in \{0, 1\}$, Modellausgabe $\hat{p} = P(Y = 1|\boldsymbol{x}) \in [0, 1]$ (oft $\hat{p} = \sigma(f(\boldsymbol{x}))$ wobei $f(\boldsymbol{x})$ der Logit ist):

$$\mathcal{L}_{\text{Log}}(y, \hat{p}) = -[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})] \tag{19}$$

2. Labels $y \in \{-1, +1\}$, Modellausgabe Score $f(\boldsymbol{x}) \in \mathbb{R}$:

$$\mathcal{L}_{\text{Log}}(y, f(\boldsymbol{x})) = \log(1 + e^{-y \cdot f(\boldsymbol{x})}) \tag{20}$$

Diese Form ist äquivalent zur ersten, wenn $\hat{p} = \sigma(f(\boldsymbol{x})) = 1/(1 + e^{-f(\boldsymbol{x})})$ und die Labels entsprechend abgebildet werden (z.B. $y_{\text{prob}} = (y_{\text{score}} + 1)/2$).

Herleitung (Maximum Likelihood): Angenommen, die bedingte Wahrscheinlichkeit des Klassenlabels folgt einer Bernoulli-Verteilung: $P(Y = y|\boldsymbol{x}) = \hat{p}^y(1 - \hat{p})^{1-y}$ für $y \in \{0, 1\}$. Gegeben sei ein Datensatz $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. Die Likelihood (Plausibilität) ist $L = \prod_{i=1}^N P(y_i|\boldsymbol{x}_i) = \prod_{i=1}^N \hat{p}_i^{y_i}(1 - \hat{p}_i)^{1-y_i}$. Die Maximierung der Likelihood ist äquivalent zur Minimierung der negativen Log-Likelihood (NLL):

$$\text{NLL} = -\log L = -\sum_{i=1}^{N} [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)] \tag{21}$$

Der Verlust für ein einzelnes Beispiel ist genau der Logistische Verlust / Binäre Kreuzentropie aus Gl. (19). Die Herleitung von Gl. (20) erfolgt durch Einsetzen von $\hat{p} = \sigma(f(\boldsymbol{x}))$ in Gl. (19) und Transformation der Labels, wie im englischen Originaltext gezeigt.

Eigenschaften und Anwendungsfälle:

Eigenschaft	Beschreibung
Vorteile	▪ Konvex und stetig differenzierbar (glatt), daher gut geeignet für gradientenbasierte Optimierung. ▪ Liefert gut kalibrierte Wahrscheinlichkeitsschätzungen (bei Verwendung mit Sigmoid/Softmax). ▪ Starke Verbindung zur Informationstheorie (Kreuzentropie). ▪ Weit verbreiteter Standard für probabilistische Klassifikationsmodelle.
Nachteile	▪ Empfindlicher gegenüber Ausreissern als der Hinge-Verlust, da er auch sehr sichere korrekte Vorhersagen ($\lvert y \cdot f(\boldsymbol{x})\rvert \gg 1$) leicht bestraft (obwohl die Strafe asymptotisch gegen null geht). ▪ Verlust geht gegen unendlich, wenn die geschätzte Wahrscheinlichkeit für eine wahre positive Klasse gegen 0 oder für eine wahre negative Klasse gegen 1 geht.
Use Cases	Training von Logistischen Regressionsmodellen, Standardwahl für binäre Klassifikationsaufgaben in neuronalen Netzen (oft gepaart mit einer finalen Sigmoid-Aktivierung).

3.4 Kategorische Kreuzentropie (Categorical Cross-Entropy)

Die Kategorische Kreuzentropie ist die Verallgemeinerung des Logistischen Verlusts auf Multiklassen-Klassifikationsprobleme ($K > 2$ Klassen).
Formel: Erfordert wahre Labels im One-Hot-kodierten Format $\boldsymbol{y} \in \{0,1\}^K$ (wobei $y_k = 1$ für die wahre Klasse k und $y_j = 0$ für $j \neq k$) und Modellausgaben als Wahrscheinlichkeitsverteilung $\hat{\boldsymbol{p}} = (\hat{p}_0, \ldots, \hat{p}_{K-1})$, wobei $\hat{p}_k = P(Y = k|\boldsymbol{x})$ und $\sum_k \hat{p}_k = 1$. Typischerweise ist $\hat{\boldsymbol{p}} = \text{softmax}(\boldsymbol{z})$, wobei $\boldsymbol{z}$ der Vektor der Logits ist.

$$\mathcal{L}_{\text{CCE}}(\boldsymbol{y}, \hat{\boldsymbol{p}}) = -\sum_{k=0}^{K-1} y_k \log(\hat{p}_k) \tag{22}$$

Da $\boldsymbol{y}$ one-hot ist, überlebt nur der Term, der der wahren Klasse c (wo $y_c = 1$) entspricht:

$$\mathcal{L}_{\text{CCE}}(\boldsymbol{y}, \hat{\boldsymbol{p}}) = -\log(\hat{p}_c) \tag{23}$$

Das bedeutet, der Verlust bestraft das Modell basierend auf der Wahrscheinlichkeit, die es der korrekten Klasse zuweist.
Herleitung (Maximum Likelihood): Angenommen, die bedingte Wahrscheinlichkeit des Klassenlabels folgt einer Multinoulli- (Kategorischen) Verteilung: $P(Y = k|\boldsymbol{x}) = \hat{p}_k$. Für eine One-Hot-kodierte Beobachtung $\boldsymbol{y}$ (mit $y_c = 1$) ist die Wahrscheinlichkeit $P(\boldsymbol{y}|\boldsymbol{x}) = \prod_{k=0}^{K-1} \hat{p}_k^{y_k} = \hat{p}_c$. Die negative Log-Likelihood für ein einzelnes Beispiel ist $-\log P(\boldsymbol{y}|\boldsymbol{x}) = -\log(\hat{p}_c)$, was genau der Kategorischen Kreuzentropie entspricht.
Eigenschaften und Anwendungsfälle:

Eigenschaft	Beschreibung
Vorteile	▪ Natürliche Erweiterung des Logistischen Verlusts für Multiklassenprobleme. ▪ Konvex (wenn auf Softmax-Ausgaben linearer Schichten angewendet) und glatt. ▪ Standard für Multiklassen-Klassifikation mit neuronalen Netzen. ▪ Liefert Wahrscheinlichkeitsverteilungen über Klassen. ▪ Starke informationstheoretische Interpretation.
Nachteile	▪ Erfordert One-Hot-kodierte Labels (oder implizite Handhabung durch Frameworks). ▪ Empfindlich gegenüber falsch gelabelten Daten (ein einzelnes falsches Label kann zu hohem Verlust führen, wenn das Modell sicher ist). ▪ Geht gegen unendlich, wenn die vorhergesagte Wahrscheinlichkeit für die wahre Klasse gegen null geht.
Use Cases	Standard-Verlustfunktion für Multiklassen-Klassifikationsprobleme, insbesondere in neuronalen Netzen (typischerweise gepaart mit einer finalen Softmax-Aktivierungsschicht).

3.5 Quadratischer Hinge-Verlust (Squared Hinge Loss)

Dies ist eine Variante des Hinge-Verlusts, bei der die Strafe quadratisch statt linear ist.
Formel: (für $y \in \{-1, +1\}$ und Score $f(\boldsymbol{x})$)

$$\mathcal{L}_{\text{SqHinge}}(y, f(\boldsymbol{x})) = \left(\max(0, 1 - y \cdot f(\boldsymbol{x}))\right)^2 \tag{24}$$

Herleitung: Eine direkte Modifikation des Standard-Hinge-Verlusts, bei der der Term, der die Margin-Verletzung darstellt, quadriert wird.
Eigenschaften und Anwendungsfälle:

Eigenschaft	Beschreibung
Vorteile	▪ Konvex und stetig differenzierbar (im Gegensatz zum Standard-Hinge-Verlust). Der Gradient ist 0 für $y \cdot f(\boldsymbol{x}) \geq 1$ und $-2(1 - y \cdot f(\boldsymbol{x})) \cdot y$ sonst. ▪ Kann manchmal aufgrund der Glattheit einfacher zu optimieren sein als der Standard-Hinge-Verlust.
Nachteile	▪ Empfindlicher gegenüber Ausreissern als der Standard-Hinge-Verlust, da Fehler quadriert werden. ▪ Weniger gebräuchlich als Standard-Hinge- oder Logistischer Verlust. Die theoretische Motivation (maximale Marge) ist direkter mit dem linearen Hinge-Verlust verbunden.
Use Cases	Eine Alternative zum Standard-Hinge-Verlust bei SVMs (manchmal als L2-SVM bezeichnet). Kann auch in anderen linearen Modellen oder neuronalen Netzen verwendet werden.

3.6 Exponentieller Verlust (Exponential Loss)

Der Exponentielle Verlust weist fehlklassifizierten Punkten eine exponentiell ansteigende Strafe basierend auf ihrem Margin-Score zu. Er ist am bekanntesten durch den AdaBoost-Algorithmus.

Formel: (für $y \in \{-1, +1\}$ und Score $f(\boldsymbol{x})$)

$$\mathcal{L}_{\text{Exp}}(y, f(\boldsymbol{x})) = e^{-y \cdot f(\boldsymbol{x})} \tag{25}$$

Herleitung: AdaBoost kann als ein vorwärts gerichteter stufenweiser additiver Modellierungsalgorithmus hergeleitet werden, der die exponentielle Verlustfunktion optimiert. In jeder Stufe wird ein schwacher Lerner hinzugefügt, um den exponentiellen Gesamtverlust des Ensembles zu minimieren.

Eigenschaften und Anwendungsfälle:

Eigenschaft	Beschreibung
Vorteile	▪ Konvex und glatt. ▪ Führt direkt zum Gewichtungsschema des AdaBoost-Algorithmus.
Nachteile	▪ Extrem empfindlich gegenüber Ausreissern und falsch gelabelten Daten aufgrund der exponentiellen Strafe. Ein einzelner Punkt mit einer grossen negativen Marge $y \cdot f(\boldsymbol{x})$ kann den Verlust dominieren. ▪ Entspricht nicht direkt Wahrscheinlichkeiten. ▪ Weniger gebräuchlich ausserhalb von Boosting-Algorithmen im Vergleich zu Logistischem oder Hinge-Verlust.
Use Cases	Hauptsächlich im Kontext von Boosting-Algorithmen verwendet, insbesondere AdaBoost.

3.7 Vergleich und Zusammenfassung

Die Wahl der richtigen Verlustfunktion hängt vom spezifischen Algorithmus, der gewünschten Ausgabe (Scores vs. Wahrscheinlichkeiten), den Anforderungen an die Robustheit und den rechnerischen Überlegungen ab. Während der Null-Eins-Verlust das wahre Klassifikationsziel darstellt, bieten Surrogat-Verluste wie Hinge-, Logistischer und Exponentieller Verlust rechentechnisch handhabbare Alternativen mit unterschiedlichen Eigenschaften. Die Kreuzentropie (Logistisch und Kategorisch) ist der Standard für probabilistische Modelle, während der Hinge-Verlust mit dem Maximum-Margin-Prinzip von SVMs verbunden ist.

Tabelle 2 fasst die Schlüsseleigenschaften und Formeln der besprochenen Verlustfunktionen zusammen. Beachten Sie, dass für die Logistische und Kategorische Kreuzentropie die Formeln mit Wahrscheinlichkeiten $(\hat{p}, \hat{\boldsymbol{p}})$ oft direkter in Implementierungen verwendet werden, die Sigmoid- oder Softmax-Aktivierungen beinhalten. Die Formeln mit dem Score $f(\boldsymbol{x})$ sind nützlich für den Vergleich mit dem Hinge- und Exponentiellen Verlust.

Tabelle 2: Vergleich von Klassifikations-Verlustfunktionen

Verlustfunktion	Formel (Gängige Form)	Konvex?	Differenzierbar?	Empfindlichkeit ggü. Ausreissern	Use Cases
Null-Eins	$\mathbb{I}[y \cdot f(\boldsymbol{x}) \leq 0]$ $(y \in \{-1, 1\})$	Nein	Nein (f.ü.)	Gering	Evaluationsmetrik
Hinge	$\max(0, 1 - y \cdot f(\boldsymbol{x}))$ $(y \in \{-1, 1\})$	Ja	Nein (bei $yf(\boldsymbol{x}) = 1$)	Mittel	SVMs
Logistisch (BCE)	$\log(1 + e^{-y \cdot f(\boldsymbol{x})})$ $(y \in \{-1, 1\})$ ODER $-[y \log \hat{p} + (1-y) \log(1-\hat{p})]$ $(y \in \{0, 1\}, \hat{p} \in [0, 1])$	Ja	Ja	Mittel-Hoch	Logistische Regression, Neuronale Netze (Binär)
Kategorische Kreuzentropie	$-\log(\hat{p}_c)$ ($\boldsymbol{y}$ one-hot, $\hat{\boldsymbol{p}}$ W'keitsvektor, c=wahre Klasse)	Ja	Ja	Mittel-Hoch	Neuronale Netze (Multiklasse)
Quadrat. Hinge	$(\max(0, 1 - y \cdot f(\boldsymbol{x})))^2$ $(y \in \{-1, 1\})$	Ja	Ja	Hoch	L2-SVMs, Alternative zu Hinge
Exponentiell	$e^{-y \cdot f(\boldsymbol{x})}$ $(y \in \{-1, 1\})$	Ja	Ja	Sehr Hoch	AdaBoost

Hinweis: $f(\boldsymbol{x})$ repräsentiert typischerweise den Rohwert (Score) oder Logit des Modells. $\hat{p}$ und $\hat{\boldsymbol{p}}$ repräsentieren vorhergesagte Wahrscheinlichkeiten. Differenzierbarkeit bezieht sich auf stetige Differenzierbarkeit. f.ü. = fast überall.

4 Kontrastive Verlustfunktionen (Contrastive Losses)

Kontrastive Verlustfunktionen sind eine zentrale Komponente des **kontrastiven Lernens**, einer Methodik, die darauf abzielt, nützliche Repräsentationen von Daten zu lernen, oft ohne explizite Labels (im Rahmen des selbst-überwachten Lernens, Self-

Supervised Learning, SSL) oder zur Verbesserung überwachter Modelle (Metric Learning). Die Grundidee besteht darin, eine Einbettungsfunktion (Encoder) f_θ zu trainieren, die Datenpunkte $\boldsymbol{x}$ in einen niedrigdimensionalen Repräsentationsraum (Embedding Space) abbildet ($\boldsymbol{h} = f_\theta(\boldsymbol{x})$), sodass ähnliche Datenpunkte nahe beieinander und unähnliche Datenpunkte weit voneinander entfernt liegen.
Dies wird erreicht, indem man für einen gegebenen **Ankerpunkt** (anchor) $\boldsymbol{h}$:

- **Positive Beispiele** $\boldsymbol{h}^+$ (z.B. andere Transformationen/Augmentationen desselben Datenpunkts, Punkte derselben Klasse) im Repräsentationsraum näher an den Anker heranzieht.
- **Negative Beispiele** $\boldsymbol{h}^-$ (z.B. Datenpunkte aus anderen Bildern/Klassen) vom Anker wegstösst.

Der "Kontrastëntsteht durch den Vergleich der Ähnlichkeit zwischen dem Anker und positiven Beispielen gegenüber der Ähnlichkeit zwischen dem Anker und negativen Beispielen. Die Formulierung des Verlusts hängt entscheidend vom gewählten **Ähnlichkeitsmass** (Similarity Measure) und der spezifischen Struktur der positiven/negativen Paare oder Tripletts ab.
Kontrastives Lernen findet breite Anwendung im selbst-überwachten Lernen für Computer Vision und NLP, im Metric Learning, in Empfehlungssystemen und bei der Gesichtserkennung.

4.1 Ähnlichkeitsmasse (Similarity Measures)

Die Wahl des Ähnlichkeitsmasses ist entscheidend dafür, wie "Näheünd "Ferneïm Einbettungsraum quantifiziert werden. Die gängigsten Masse sind:

- **Kosinus-Ähnlichkeit (Cosine Similarity):** Misst den Kosinus des Winkels zwischen zwei Vektoren $\boldsymbol{u}$ und $\boldsymbol{v}$. Sie ist unempfindlich gegenüber der Magnitude der Vektoren und konzentriert sich auf die Orientierung. Werte liegen im Bereich $[-1, 1]$, wobei 1 perfekte Übereinstimmung, -1 entgegengesetzte Richtung und 0 Orthogonalität bedeutet. Oft verwendet für hochdimensionale Daten (wie Text-Embeddings oder Bild-Features) und typischerweise in Verbindung mit normalisierten Embeddings ($\|\boldsymbol{h}\|_2 = 1$).

$$\mathsf{sim}_{\mathsf{cos}}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\|_2 \|\boldsymbol{v}\|_2} \tag{26}$$

- **Euklidischer Abstand (L_2-Distanz):** Misst den geradlinigen Abstand zwischen zwei Punkten im Raum. Werte liegen im Bereich $[0, \infty)$. Im Gegensatz zur Kosinus-Ähnlichkeit ist er empfindlich gegenüber der Magnitude. Kontrastive Verluste, die auf Distanz basieren, zielen darauf ab, die Distanz für positive Paare zu *minimieren* und für negative Paare zu *maximieren* (oft über eine Marge hinaus). Um ihn als Ähnlichkeitsmass zu interpretieren, kann eine invertierende Transformation verwendet werden (z.B. $\exp(-d^2)$).

$$d_{\text{euc}}(\boldsymbol{u}, \boldsymbol{v}) = \|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{\sum_i (u_i - v_i)^2} \tag{27}$$

- **Skalarprodukt (Dot Product):** Das einfache Skalarprodukt $\boldsymbol{u} \cdot \boldsymbol{v}$ kann ebenfalls als Ähnlichkeitsmass dienen. Es ist jedoch stark von den Vektormagnituden abhängig. Wenn die Vektoren auf eine Einheitskugel normiert sind ($\|\boldsymbol{u}\|_2 = \|\boldsymbol{v}\|_2 = 1$), ist das Skalarprodukt äquivalent zur Kosinus-Ähnlichkeit.

Die Wahl des Masses beeinflusst die Geometrie des erlernten Repräsentationsraums und die Formulierung der Verlustfunktion.

Übersicht der Ähnlichkeits-/Distanzmasse:

Mass	Formel	Wertebereich	Typische Verwendung (Kontrastives Lernen)
Kosinus-Ähnlichkeit	$\frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\| \|\boldsymbol{v}\|}$	$[-1, 1]$	InfoNCE/NT-Xent, SSL, hohe Dimensionen
Euklidischer Abstand (L_2)	$\|\boldsymbol{u} - \boldsymbol{v}\|_2$	$[0, \infty)$	Contrastive Loss (Paar), Triplet Loss, Metric Learning
Skalarprodukt	$\boldsymbol{u} \cdot \boldsymbol{v}$	$(-\infty, \infty)$	Ähnlich zu Kosinus bei normierten Vektoren

4.2 Contrastive Loss (Paar-basiert)

Dies ist eine der frühesten Formulierungen kontrastiven Lernens, oft verwendet in Siamesischen Netzwerken (Hadsell et al., 2006). Der Verlust wird separat für positive und negative Paare definiert.

Erklärung: Für ein Paar von Eingaben $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ und deren Embeddings $(\boldsymbol{h}_1, \boldsymbol{h}_2)$ wird ein Label y verwendet ($y = 1$ für ein positives Paar, $y = 0$ für ein negatives Paar). Das Ziel ist, die Distanz $d = d(\boldsymbol{h}_1, \boldsymbol{h}_2)$ für positive Paare klein zu halten und für negative

Paare sicherzustellen, dass sie grösser als eine definierte Marge m ist. Typischerweise wird der Euklidische Abstand verwendet.
Formel: Der Verlust für einen Datensatz von N Paaren ist:

$$\mathcal{L}_{\text{Contrastive}} = \frac{1}{N} \sum_{i=1}^{N} \left[y_i d_i^2 + (1 - y_i) \max(0, m - d_i)^2 \right] \tag{28}$$

Hier ist $d_i = d_{\text{euc}}(\boldsymbol{h}_{i,1}, \boldsymbol{h}_{i,2})$ die Distanz des i-ten Paares, $y_i \in \{0, 1\}$ das Label des Paares, und $m > 0$ die Marge. (Manchmal wird d_i statt d_i^2 verwendet).
Motivation: Die Formel ist intuitiv:

- Wenn $y_i = 1$ (positives Paar), ist der Verlust d_i^2. Die Minimierung dieses Terms zieht positive Paare zusammen.
- Wenn $y_i = 0$ (negatives Paar), ist der Verlust $\max(0, m - d_i)^2$. Dieser Term ist nur dann grösser als null, wenn die Distanz d_i kleiner als die Marge m ist. Die Minimierung bestraft also negative Paare, die zu nah beieinander liegen, und drängt sie auseinander, bis ihre Distanz mindestens m beträgt.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Minimiere Distanz für positive Paare, maximiere sie über eine Marge m für negative Paare.
Typisches Ähnlichkeitsmass	Euklidischer Abstand (L_2).
Formel (pro Paar)	$yd^2 + (1 - y) \max(0, m - d)^2$.
Vorteile	▪ Einfache und intuitive Formulierung. ▪ Effektiv für Metric Learning und Verifikationsaufgaben.
Nachteile/ Herausforderungen	▪ Erfordert die Definition von positiven und negativen Paaren (ggf. Labels nötig oder aufwändige Generierung). ▪ Leistung hängt von der Wahl der Marge m ab. ▪ Schwierigkeiten bei der Auswahl informativer negativer Paare (Sampling).
Use Cases	Metric Learning, Gesichtserkennung/-verifikation, Signaturverifikation, Training Siamesischer Netzwerke.

4.3 Triplet Loss

Der Triplet Loss (Weinberger et al., 2006; Schroff et al., 2015 - FaceNet) verwendet statt Paaren sogenannte Tripletts, bestehend aus einem Anker-, einem positiven und einem negativen Beispiel.

Erklärung: Für jedes Triplett $(\boldsymbol{x}_a, \boldsymbol{x}_p, \boldsymbol{x}_n)$ mit Embeddings $(\boldsymbol{h}_a, \boldsymbol{h}_p, \boldsymbol{h}_n)$ soll der Abstand zwischen Anker und Positivem $d(a, p)$ kleiner sein als der Abstand zwischen Anker und Negativem $d(a, n)$, und zwar um eine Marge m.
Formel: Der Verlust über N Tripletts ist:

$$\mathcal{L}_{\text{Triplet}} = \frac{1}{N} \sum_{i=1}^{N} \max(0, d(\boldsymbol{h}_{a,i}, \boldsymbol{h}_{p,i})^2 - d(\boldsymbol{h}_{a,i}, \boldsymbol{h}_{n,i})^2 + m) \tag{29}$$

Auch hier wird oft der Euklidische Abstand verwendet, und manchmal werden die Distanzen nicht quadriert. $m > 0$ ist die Marge.
Motivation: Der Verlustterm ist nur dann positiv, wenn $d(a, n)^2 < d(a, p)^2 + m$. Die Minimierung des Verlusts erzwingt also $d(a, p)^2 + m \leq d(a, n)^2$. Dies stellt sicher, dass der Anker dem positiven Beispiel signifikant näher ist als dem negativen Beispiel.
Triplet Mining: Eine grosse Herausforderung ist die Auswahl von informativen Tripletts. Zufällige Tripletts führen oft zu einem Verlust von null (wenn die Bedingung bereits erfüllt ist) und somit zu langsamer Konvergenz. Strategien wie "Hard Negative Mining"(Auswahl von negativen Beispielen, die der Marge am nächsten kommen oder sie verletzen) sind entscheidend für den Erfolg.
Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Erzwinge, dass der Anker dem positiven Beispiel um eine Marge m näher ist als dem negativen Beispiel.
Typisches Ähnlichkeitsmass	Euklidischer Abstand (L_2).
Formel (pro Triplett)	$\max(0, d(a, p)^2 - d(a, n)^2 + m)$.
Vorteile	▪ Lernt eine relative Ähnlichkeitsstruktur. ▪ Sehr erfolgreich im Metric Learning, insbesondere Gesichtserkennung (FaceNet).
Nachteile/ Herausforderungen	▪ Erfordert die Bildung von Tripletts. ▪ Kritische Abhängigkeit von der Strategie zur Auswahl der Tripletts (Triplet Mining). ▪ Batch-Grösse und Sampling können komplex sein. ▪ Wahl der Marge m.
Use Cases	Gesichtserkennung, Person Re-Identification, Bildsuche, Metric Learning im Allgemeinen.

4.4 InfoNCE / NT-Xent Loss

InfoNCE (Information Noise Contrastive Estimation) ist ein moderner kontrastiver Verlust, der insbesondere im selbst-überwachten Lernen (SSL) sehr erfolgreich ist (z.B. in CPC, SimCLR, MoCo). NT-Xent (Normalized Temperature-scaled Cross Entropy) ist eine spezifische Implementierung davon, die in SimCLR verwendet wird.

Erklärung: Die Kernidee ist, das kontrastive Lernen als ein Klassifikationsproblem zu formulieren: Für einen Anker $\boldsymbol{h}_i$ soll sein positives Beispiel $\boldsymbol{h}_{i^+}$ aus einer Menge von K negativen Beispielen $\{\boldsymbol{h}_k^-\}$ korrekt identifiziert werden. Dies basiert auf der Maximierung der unteren Schranke der gegenseitigen Information (Mutual Information) zwischen verschiedenen SSichten"(z.B. Augmentationen) desselben Datenpunkts. Typischerweise werden Kosinus-Ähnlichkeit und ein Temperatur-Skalierungsfaktor τ verwendet.

Formel (InfoNCE): Für einen Anker $\boldsymbol{h}_i$, sein positives Beispiel $\boldsymbol{h}_{i^+}$ und K negative Beispiele $\{\boldsymbol{h}_k^-\}_{k=1}^K$:

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{\exp(\text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_{i^+})/\tau)}{\exp(\text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_{i^+})/\tau) + \sum_{k=1}^K \exp(\text{sim}(\boldsymbol{h}_i, \boldsymbol{h}_k^-)/\tau)}\right] \tag{30}$$

Dies hat die Form einer Softmax-Kreuzentropie, wobei die Logits durch die skalierten Ähnlichkeiten gegeben sind. sim ist typischerweise die Kosinus-Ähnlichkeit.

Formel (NT-Xent - SimCLR Variante): In SimCLR werden für jedes Bild $\boldsymbol{x}$ in einem Batch der Grösse N zwei augmentierte Versionen erzeugt, was zu $2N$ Embeddings $(\boldsymbol{h}_1, ..., \boldsymbol{h}_{2N})$ führt. Für ein positives Paar $(\boldsymbol{h}_i, \boldsymbol{h}_j)$ (die von demselben Originalbild stammen) werden alle anderen $2(N-1)$ Embeddings im Batch als negative Beispiele betrachtet. Der Verlust für das Paar (i, j) ist:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}_{\cos}(\boldsymbol{h}_i, \boldsymbol{h}_j)/\tau)}{\sum_{k=1, k\neq i}^{2N} \exp(\text{sim}_{\cos}(\boldsymbol{h}_i, \boldsymbol{h}_k)/\tau)} \tag{31}$$

Der Gesamtverlust ist der Durchschnitt von $\ell_{i,j} + \ell_{j,i}$ über alle positiven Paare (i, j) im Batch.

Temperatur τ: Der Temperaturparameter τ (typischerweise ein kleiner Wert wie 0.1 oder 0.07) skaliert die Ähnlichkeiten vor der Softmax-Funktion. Eine niedrige Temperatur erhöht die Konzentration der Verteilung und gewichtet "harte"negative Beispiele (solche, die dem Anker ähnlich sind) stärker.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Grundidee	Identifiziere das positive Beispiel unter vielen negativen Beispielen (Klassifikations-Analogie). Maximiert Mutual Information.
Typisches Ähnlichkeitsmass	Kosinus-Ähnlichkeit.
Formel (InfoNCE)	Softmax-Kreuzentropie über skalierte Ähnlichkeiten. Siehe Gl. (30).
Wichtige Aspekte	▪ Temperaturparameter τ zur Skalierung. ▪ Benötigt eine grosse Anzahl negativer Beispiele für gute Leistung (oft aus demselben Batch oder einer Memory Bank). ▪ Normalisierung der Embeddings oft vorteilhaft.
Vorteile	▪ State-of-the-Art Ergebnisse im selbst-überwachten Lernen. ▪ Skaliert gut mit grosser Anzahl negativer Beispiele. ▪ Theoretische Verbindung zur Mutual Information.
Nachteile/ Herausforderungen	▪ Erfordert oft grosse Batch-Grössen oder spezielle Techniken (z.B. Memory Banks) für viele Negative. ▪ Leistung kann sensitiv auf die Wahl der Temperatur τ und der Datenaugmentierungen sein. ▪ Bias durch Sampling von Negativen innerhalb des Batches möglich (SSampling Bias").
Use Cases	Selbst-überwachtes Vorlernen von visuellen und sprachlichen Repräsentationen (SimCLR, MoCo, CPC, etc.).

4.5 Vergleich Kontrastiver Verlustfunktionen

Kontrastive Verlustfunktionen bieten flexible Werkzeuge zum Lernen von Repräsentationen durch Vergleich. Die Wahl der Funktion hängt von der Aufgabe und den verfügbaren Daten ab. Die paar-basierte Contrastive Loss und die Triplet Loss sind oft im Metric Learning und bei Verifikationsaufgaben zu finden, wo explizite positive/-negative Beziehungen (oft durch Labels) definiert sind oder leicht abgeleitet werden können; sie erfordern jedoch sorgfältiges Sampling oder Mining. InfoNCE/NT-Xent dominiert im modernen selbst-überwachten Lernen, wo positive Paare durch Datenaugmentation erzeugt werden und eine grosse Menge an negativen Beispielen (oft der Rest des Batches) verwendet wird, um robuste, allgemeine Repräsentationen zu lernen. Die Wahl des Ähnlichkeitsmasses ist ebenfalls entscheidend, wobei Kosinus-Ähnlichkeit bei InfoNCE und Euklidischer Abstand bei den älteren Methoden vorherrschen.
Tabelle 3 fasst die Hauptunterschiede zusammen.

Tabelle 3: Vergleich von Kontrastiven Verlustfunktionen

Verlustfunktion	Grundidee	Ähnlichkeitsmass (typ.)	Benötigt	Hauptvorteil	Hauptherausforderung
Contrastive Loss (Paar)	Nähe für Pos., Ferne ($> m$) für Neg.	Eukl. Distanz (L_2)	Positive/Negative Paare, Marge m	Intuitiv, gut für Verifikation	Sampling von Paaren, Marge m
Triplet Loss	Anker näher an Pos. als an Neg. (mit Marge m)	Eukl. Distanz (L_2)	Tripletts (a, p, n), Marge m	Lernt relative Ähnlichkeit	Triplet Mining, Marge m
InfoNCE / NT-Xent	Identifiziere Pos. unter vielen Neg. (Klassifikation)	Kosinus-Ähnlichkeit	Pos. Paar, viele Negative, Temperatur τ	State-of-the-Art SSL, skaliert gut	Grosse Batches/Memory Bank, τ-Wahl

Hinweis: SSL = Self-Supervised Learning. m = Marge, τ = Temperatur.

5 Adversariale Verlustfunktionen (Adversarial Losses)

Adversariale Verlustfunktionen sind das Herzstück von Generative Adversarial Networks (GANs), einem populären Ansatz im Bereich der generativen Modellierung. GANs bestehen typischerweise aus zwei Komponenten, die in einem Minimax-Spiel gegeneinander antreten:

- **Generator (G):** Versucht, Daten zu erzeugen (z.B. Bilder, Texte), die von echten Daten nicht zu unterscheiden sind. Er nimmt einen Zufallsvektor $\boldsymbol{z}$ aus einem Prior-Raum (z.B. einer Normalverteilung p_z) als Eingabe und erzeugt eine synthetische Probe $G(\boldsymbol{z})$. Das Ziel ist es, die Verteilung p_g der generierten Daten so zu formen, dass sie der Verteilung p_{data} der echten Daten $\boldsymbol{x}$ möglichst ähnlich ist.
- **Diskriminator (D):** Versucht zu entscheiden, ob eine gegebene Datenprobe echt (aus p_{data}) oder künstlich (aus p_g, also von G erzeugt) ist. Er gibt typischerweise einen Wert aus, der die Wahrscheinlichkeit (oder einen Score) repräsentiert, dass die Eingabe echt ist.

Der ädversariale Verlustërgibt sich aus diesem kompetitiven Prozess. Der Diskriminator wird trainiert, um echte und künstliche Proben korrekt zu klassifizieren, während der Generator trainiert wird, um Proben zu erzeugen, die den Diskriminator täuschen. Dieses dynamische Gleichgewicht führt im Idealfall dazu, dass der Generator lernt, realistische Daten zu erzeugen.
Verschiedene Formulierungen des adversarialen Verlusts wurden vorgeschlagen, um unterschiedliche Distanzmasse zwischen p_{data} und p_g zu optimieren oder um häufig auftretende Trainingsprobleme wie Modenkollaps (Mode Collapse) oder verschwindende Gradienten (Vanishing Gradients) zu mildern.
Wir verwenden die folgende Notation: $\boldsymbol{x} \sim p_{data}$ ist eine echte Datenprobe, $\boldsymbol{z} \sim p_z$ ist ein Rauschvektor, $G(\boldsymbol{z})$ ist eine generierte (künstliche) Probe, $D(\boldsymbol{x})$ ist die Ausgabe

des Diskriminators für eine echte Probe, und $D(G(\boldsymbol{z}))$ ist die Ausgabe des Diskriminators für eine künstliche Probe. $\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\cdot]$ bezeichnet den Erwartungswert über die echte Datenverteilung und $\mathbb{E}_{\boldsymbol{z}\sim p_z}[\cdot]$ den Erwartungswert über die Prior-Verteilung des Rauschens.

5.1 Minimax-Verlust (Original GAN)

Der ursprüngliche GAN-Verlust, vorgeschlagen von Goodfellow et al. (2014), basiert auf einem Minimax-Spiel, das theoretisch die Jensen-Shannon-Divergenz (JSD) zwischen der echten Datenverteilung p_{data} und der Generatorverteilung p_g minimiert.
Formel (Minimax-Ziel): Das Ziel ist es, das folgende Minimax-Problem zu lösen:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(1 - D(G(\boldsymbol{z})))] \tag{32}$$

Hier wird angenommen, dass $D(\cdot)$ die Wahrscheinlichkeit ausgibt, dass die Eingabe echt ist ($D(\cdot) \in [0,1]$, typischerweise über eine Sigmoid-Aktivierung).
Herleitung/Motivation: Die Zielfunktion $V(D,G)$ entspricht der binären Kreuzentropie für einen Klassifikator D, der echte Daten (Label 1) von künstlichen Daten (Label 0) unterscheiden soll. Bei optimalem Diskriminator $D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}$ reduziert sich das Minimax-Problem zu $\min_G(2 \cdot JSD(p_{data}||p_g) - 2\log 2)$. Die Minimierung bezüglich G minimiert also die JSD zwischen der echten und der generierten Verteilung.
Separate Verluste für das Training: In der Praxis werden G und D abwechselnd trainiert, wobei separate Verlustfunktionen minimiert werden:

- **Diskriminator-Training:** Maximiere $V(D,G)$ bezüglich D. Dies ist äquivalent zur Minimierung des negativen $V(D,G)$, was einer Standard-Kreuzentropie-Verlustfunktion entspricht:

$$\mathcal{L}_D = -\left(\mathbb{E}_{\boldsymbol{x}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))]\right) \tag{33}$$

- **Generator-Training (Original):** Minimiere $V(D,G)$ bezüglich G. Dies entspricht der Minimierung von $\mathcal{L}_G^{\text{orig}} = \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))]$. Dieses Ziel leidet jedoch unter dem Problem der *saturierenden Gradienten*: Wenn der Diskriminator die künstlichen Proben sehr gut erkennt ($D(G(\boldsymbol{z})) \approx 0$), wird der Gradient

von $\log(1 - D(G(\boldsymbol{z})))$ bezüglich der Parameter von G sehr klein, was das Lernen verlangsamt oder stoppt.

- **Generator-Training (Non-Saturating Heuristik):** Um das Sättigungsproblem zu umgehen, wird in der Praxis oft ein alternatives Ziel für G verwendet: Maximiere $\mathbb{E}_{\boldsymbol{z}}[\log D(G(\boldsymbol{z}))]$, was äquivalent zur Minimierung von

$$\mathcal{L}_G^{\text{ns}} = -\mathbb{E}_{\boldsymbol{z}}[\log D(G(\boldsymbol{z}))] \tag{34}$$

ist. Dieses Ziel liefert stärkere Gradienten, besonders zu Beginn des Trainings.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Ziel (theoretisch)	Minimierung der Jensen-Shannon-Divergenz (JSD) zwischen p_{data} und p_g.
Diskriminator-Verlust $\mathcal{L}_D$	Standard Binäre Kreuzentropie (BCE). Siehe Gl. (33).
Generator-Verlust $\mathcal{L}_G$ (non-saturating)	Modifizierte BCE, um Gradientensättigung zu vermeiden. Siehe Gl. (34).
Vorteile	▪ Klare theoretische Fundierung (JSD-Minimierung). ▪ Einfache Implementierung mit Standard-Kreuzentropie.
Probleme/ Herausforderungen	▪ Trainingsinstabilitäten (z.B. Modenkollaps, verschwindende Gradienten). ▪ Schwierigkeiten bei der Konvergenz, erfordert sorgfältige Abstimmung der Hyperparameter. ▪ JSD ist problematisch bei disjunkten Verteilungen. ▪ Non-saturating $\mathcal{L}_G$ entspricht nicht mehr direkt dem ursprünglichen Minimax-Spiel.
Use Cases	Grundlage für viele frühe GAN-Architekturen. Wird oft als Basis oder zum Vergleich herangezogen.

5.2 Wasserstein-Verlust (WGAN & WGAN-GP)

Der Wasserstein-GAN (WGAN)-Verlust, vorgeschlagen von Arjovsky et al. (2017), zielt darauf ab, die Trainingsstabilität von GANs zu verbessern, indem statt der JSD die Wasserstein-1-Distanz (auch Earth Mover's Distance, EMD) minimiert wird. Die W-Distanz hat auch bei disjunkten Verteilungen aussagekräftigere Gradienten.
Formel (Wasserstein-1-Distanz): Die W-1-Distanz zwischen p_{data} und p_g ist definiert als:

$$W(p_{data}, p_g) = \sup_{\|f\|_L \leq 1} (\mathbb{E}_{\boldsymbol{x} \sim p_{data}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z} \sim p_z}[f(G(\boldsymbol{z}))]) \tag{35}$$

wobei das Supremum über alle 1-Lipschitz-Funktionen f genommen wird. Im WGAN-Kontext wird die Funktion f durch den *Kritiker* (Critic, C) approximiert, der an die Stelle des Diskriminators tritt. Der Kritiker gibt einen unbeschränkten Score aus, keine Wahrscheinlichkeit.

Verlustfunktionen:

- **Kritiker-Training:** Der Kritiker C wird trainiert, um den Ausdruck in Gl. (35) zu maximieren. Dies entspricht der Minimierung von:

$$\mathcal{L}_C = -\left(\mathbb{E}_{\boldsymbol{x}}[C(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}}[C(G(\boldsymbol{z}))]\right) \tag{36}$$

- **Generator-Training:** Der Generator G wird trainiert, um die W-Distanz zu minimieren. Da $\mathbb{E}_{\boldsymbol{x}}[C(\boldsymbol{x})]$ nicht von G abhängt, entspricht dies der Maximierung von $\mathbb{E}_{\boldsymbol{z}}[C(G(\boldsymbol{z}))]$, oder der Minimierung von:

$$\mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}}[C(G(\boldsymbol{z}))] \tag{37}$$

Durchsetzung der Lipschitz-Bedingung: Die grösste Herausforderung bei WGANs ist die Sicherstellung, dass der Kritiker C (approximativ) 1-Lipschitz bleibt.

- **WGAN (Weight Clipping):** Die ursprüngliche Methode beschränkt die Gewichte des Kritikers auf einen kleinen Bereich (z.B. $[-0.01, 0.01]$). Dies ist einfach, kann aber zu Optimierungsproblemen oder reduzierter Kapazität des Kritikers führen.

- **WGAN-GP (Gradient Penalty):** Gulrajani et al. (2017) schlugen vor, der Kritiker-Verlustfunktion einen Strafterm hinzuzufügen, der Abweichungen des Gradientennormen von 1 bestraft:

$$\mathcal{L}_{GP} = \lambda \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{x}}}[(\|\nabla_{\hat{\boldsymbol{x}}} C(\hat{\boldsymbol{x}})\|_2 - 1)^2] \tag{38}$$

  Hierbei ist $\hat{\boldsymbol{x}}$ eine Stichprobe, die zufällig zwischen einer echten Probe $\boldsymbol{x}$ und einer künstlichen Probe $G(\boldsymbol{z})$ interpoliert wird ($p_{\hat{x}}$ ist die Verteilung dieser interpolierten Punkte), und λ ist ein Hyperparameter (oft $\lambda = 10$). $\mathcal{L}_C^{\text{WGAN-GP}} = \mathcal{L}_C + \mathcal{L}_{GP}$. Diese Methode ist stabiler und führt oft zu besseren Ergebnissen.

Eigenschaften und Anforderungen:

Eigenschaft	Beschreibung
Ziel (theoretisch)	Minimierung der Wasserstein-1-Distanz zwischen p_{data} und p_g.
Kritiker-Verlust $\mathcal{L}_C$ (WGAN-GP)	$\mathcal{L}_C = -(\mathbb{E}_{\boldsymbol{x}}[C(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}}[C(G(\boldsymbol{z}))]) + \lambda \mathbb{E}_{\hat{\boldsymbol{x}}}[(\|\nabla_{\hat{\boldsymbol{x}}} C(\hat{\boldsymbol{x}})\|_2 - 1)^2]$.
Generator-Verlust $\mathcal{L}_G$	$\mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}}[C(G(\boldsymbol{z}))]$.
Vorteile	▪ Deutlich verbesserte Trainingsstabilität im Vergleich zum originalen GAN. ▪ Weniger anfällig für Modenkollaps. ▪ Der Kritiker-Verlust korreliert oft mit der Bildqualität (nützlich für Monitoring). ▪ Theoretisch fundierte Gradienten auch bei disjunkten Verteilungen.
Nachteile/ Anforderungen	▪ Erfordert die Durchsetzung der Lipschitz-Bedingung (Weight Clipping problematisch, Gradient Penalty rechenintensiver). ▪ Konvergenz kann langsamer sein als bei Standard-GANs. ▪ Kritiker-Output ist unbeschränkt (Score), keine Wahrscheinlichkeit.
Use Cases	Sehr populär für Bildgenerierung und andere generative Aufgaben, bei denen Stabilität wichtig ist. Basis für viele fortgeschrittene GANs.

5.3 Least Squares Verlust (LSGAN)

Der Least Squares GAN (LSGAN), vorgeschlagen von Mao et al. (2017), ersetzt die Sigmoid-Kreuzentropie-Verluste des originalen GAN durch Least-Squares-(Quadratmittel)-Verluste.

Formel: Der Diskriminator D (der hier wieder unbeschränkte Scores ausgibt) und der Generator G minimieren folgende Verlustfunktionen, wobei a, b, c Zielwerte sind:

$$\mathcal{L}_D^{\text{LSGAN}} = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}}[(D(\boldsymbol{x}) - b)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}}[(D(G(\boldsymbol{z})) - a)^2] \tag{39}$$

$$\mathcal{L}_G^{\text{LSGAN}} = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}}[(D(G(\boldsymbol{z})) - c)^2] \tag{40}$$

Eine übliche Wahl der Parameter ist $a = 0, b = 1, c = 1$ (oder alternativ $a = -1, b = 1, c = 1$). Mit $a = 0, b = 1$ versucht der Diskriminator, echte Proben auf 1 und künstliche auf 0 zu mappen. Mit $c = 1$ versucht der Generator, den Diskriminator dazu zu bringen, seine künstlichen Proben als 1 zu klassifizieren.

Motivation: Die Verwendung des quadratischen Fehlers bestraft Proben, die zwar auf der korrekten Seite der Entscheidungsgrenze liegen, aber weit davon entfernt sind. Dies kann zu stabileren Gradienten führen als die Sigmoid-Kreuzentropie, die für ßu einfach"klassifizierte Proben sättigt (Gradient wird klein). LSGAN zielt darauf ab, die künstlichen Daten näher an die Entscheidungsgrenze zu ßiehen", die durch die echten Daten definiert ist.

Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Ziel	Minimierung eines Pearson χ^2-Divergenz-ähnlichen Ziels (implizit). Stabilisierung des Trainings durch Vermeidung von Gradientensättigung.
Diskriminator-Verlust $\mathcal{L}_D$	Quadratischer Fehler zu Zielwerten a (fake) und b (real). Siehe Gl. (39).
Generator-Verlust $\mathcal{L}_G$	Quadratischer Fehler zum Zielwert c (oft derselbe wie b). Siehe Gl. (40).
Vorteile	▪ Stabilere Gradienten im Vergleich zum originalen GAN mit Sigmoid-Kreuzentropie. ▪ Oft schnellere Konvergenz und bessere Ergebnisqualität als originaler GAN. ▪ Einfache Implementierung.
Probleme/ Herausforderungen	▪ Kann immer noch unter Modenkollaps leiden. ▪ Die Wahl der Zielwerte a, b, c kann die Leistung beeinflussen. ▪ Weniger theoretisch fundiert als WGAN bezüglich der optimierten Distanz.
Use Cases	Weit verbreitete Alternative zum originalen GAN-Verlust, besonders bei Bildgenerierungsaufgaben.

5.4 Hinge-Verlust (Adversarial Hinge Loss)

Eine weitere populäre Alternative, die oft in modernen GANs wie SAGAN oder BigGAN verwendet wird, ist die Adaption des Hinge-Verlusts für das adversariale Training.
Formel (Gängige Variante): Der Diskriminator D (der unbeschränkte Scores ausgibt) und der Generator G minimieren folgende Hinge-basierte Verluste:

$$\mathcal{L}_D^{\text{Hinge}} = \mathbb{E}_{\boldsymbol{x}}[\max(0, 1 - D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}}[\max(0, 1 + D(G(\boldsymbol{z})))] \tag{41}$$

$$\mathcal{L}_G^{\text{Hinge}} = -\mathbb{E}_{\boldsymbol{z}}[D(G(\boldsymbol{z}))] \tag{42}$$

Hierbei versucht der Diskriminator, echte Proben auf einen Score von ≥ 1 und künstliche Proben auf einen Score von ≤ -1 zu bringen. Der Generator versucht, die Scores seiner künstlichen Proben zu maximieren (also $\mathcal{L}_G$ zu minimieren).
Motivation: Ähnlich wie der Standard-Hinge-Verlust in der Klassifikation, zielt diese Formulierung auf eine maximale Marge zwischen den Scores für echte und künstliche Daten ab. Sie bestraft nur Scores, die die Marge verletzen. Dies hat sich empirisch als sehr effektiv für stabiles Training und hohe Ergebnisqualität erwiesen.
Eigenschaften und Herausforderungen:

Eigenschaft	Beschreibung
Ziel	Maximierung der Marge zwischen den Scores für echte und künstliche Daten.
Diskriminator-Verlust $\mathcal{L}_D$	Summe zweier Hinge-Terme für echte (≥ 1) und künstliche (≤ -1) Samples. Siehe Gl. (41).
Generator-Verlust $\mathcal{L}_G$	Maximierung des Diskriminator-Scores für künstliche Samples. Siehe Gl. (42).
Vorteile	▪ Empirisch sehr gute Leistung und Trainingsstabilität. ▪ Weniger empfindlich gegenüber Ausreissern als quadratische Verluste. ▪ Einfache Implementierung.
Probleme/ Herausforderungen	▪ Weniger direkte theoretische Interpretation der optimierten Divergenz im Vergleich zu original GAN oder WGAN. ▪ Kann, wie andere GANs, immer noch Moden vernachlässigen.
Use Cases	Standardwahl in vielen modernen, hochleistungsfähigen GAN-Architekturen (z.B. SAGAN, BigGAN) für Bildsynthese.

5.5 Vergleich Adversarialer Verlustfunktionen

Die Wahl der adversarialen Verlustfunktion hat erheblichen Einfluss auf die Stabilität des Trainingsprozesses und die Qualität der generierten Ergebnisse. Während der originale Minimax-Verlust eine klare theoretische Grundlage hat (JSD-Minimierung), leidet er oft unter praktischen Problemen. WGANs bieten eine verbesserte theoretische Fundierung (Wasserstein-Distanz) und empirische Stabilität, erfordern aber die Handhabung der Lipschitz-Bedingung. LSGAN und der adversariale Hinge-Verlust sind pragmatische Alternativen, die oft gute Stabilität und Leistung durch Modifikation der Zielfunktion erreichen, um Gradientenprobleme zu vermeiden. Die Wahl hängt oft von der spezifischen Anwendung, der Architektur und den verfügbaren Rechenressourcen ab.
Tabelle 4 bietet einen zusammenfassenden Überblick.

Tabelle 4: Vergleich von Adversarialen Verlustfunktionen

Verlustfunktion	Ziel (Distanz/Div.)	D-Output	G-Verlust (typisch, min.)	Hauptvorteil	Hauptherausforderung
Original GAN (Minimax)	JSD (theor.)	W'keit $[0, 1]$	$-\mathbb{E}_{\boldsymbol{z}}[\log D(G(\boldsymbol{z}))]$	Theor. Fundierung	Instabilität, Vanishing Gradients
WGAN-GP	Wasserstein-1	Score $\mathbb{R}$	$-\mathbb{E}_{\boldsymbol{z}}[C(G(\boldsymbol{z}))]$	Stabilität, Korrelation mit Qualität	Lipschitz (Gradient Penalty)
LSGAN	Pearson χ^2-ähnlich	Score $\mathbb{R}$	$\frac{1}{2}\mathbb{E}_{\boldsymbol{z}}[(D(G(\boldsymbol{z})) - 1)^2]$	Stabilität ggü. Original-GAN	Weniger theor. fundiert als WGAN
Adversarial Hinge	Margin Maximierung	Score $\mathbb{R}$	$-\mathbb{E}_{\boldsymbol{z}}[D(G(\boldsymbol{z}))]$	Empirisch hohe Leistung & Stabilität	Weniger klare Divergenz-Interpretation

Hinweis: JSD = Jensen-Shannon Divergence. D = Diskriminator, C = Kritiker, G = Generator. G-Verluste sind zur Minimierung dargestellt.

Lösungen