Cheatsheet DL: Hyperparameter und sonstige Trainingskonfigurationen

(Disclaimer: Kein Anspruch auf Vollständigkeit. Nicht zitierbar. Recherchiert! Experimentiert! Sucht euch Literatur!)

→ Batch Size:

ⓘ	→ Längere Rechenzeit pro Iteration → Braucht mehr Ressourcen → Geringere Varianz, d.h. kann weniger Kontext erfassen
ⓘ	→ Schnellere Aktualisierung der Gewichtungen → Braucht weniger Ressourcen, ist aber u. U. weniger effizient → Erhöhte Varianz, d.h. besser bei unbalancierten Datensets; kann mehr Kontext erfassen
Empfehlung	→ Experimentieren, um Balance zwischen Auslastung der Ressourcen und Modelleistung zu schaffen

→ Epochenanzahl

ⓘ	→ Längere Trainingszeit → Gefahr von Overfitting → Gründlicheres Training, längere Modellanpassung
ⓘ	→ Kürzere Trainingszeit → Gefahr von Underfitting → Beenden des Trainings vor Erreichen des Loss-Minimums
Empfehlung	→ Experimentieren, um Balance zwischen Under- und Overfitting zu schaffen

→ Learning Rate

ⓘ	→ Kürzere Trainingszeit, schnelle Konvergenz (Stabilität in der Anpassung der Gewichtungen) → Gefahr von Divergenz (Schwankungen in der Anpassung der Gewichtungen)
ⓘ	→ Stabileres Training, wenig Divergenz → Längeres Training, Konvergenz dauert länger
Empfehlung	→ Experimentieren und anhand der Validierungswerte die Learning Rate anpassen

	→ Siehe Learning Rate Scheduler (bspw. Adam oder RMSprop)

→ Aktivierungsfunktion

Es folgen drei Beispiele von Aktivierungsfunktionen für Klassifikationstasks. Recherchiert u.a. hier: https://keras.io/api/layers/activations/.

Wahl ist abhängig von u.a. Task, Netzarchitektur und Charakteristik der Daten.

Sigmoid Function	→ Ausgabe zwischen 0 und 1; eindeutige Ergebnisse für binäre Klassifikation → Stabil im Training → Anfällig für „Vanishing Gradient"-Problem → Schlechte Performanz bei tiefen Netzen
ReLU (Rectified Linear Unit)	→ Schnelle Konvergenz → Weniger anfällig für „Vanishing Gradient"-Problem → Bei zu tiefen Netzen hat auch ReLU das „Vanishing Gradient"-Problem
Softmax	→ Liefert eine Wahrscheinlichkeitsverteilung, die sich zu 1 aufsummiert; geeignet für Multilabel Klassifikation → Große Inputwerte können zu instabilen Berechnungen des Outputs führen

→ Lossfunktion

Es folgen zwei Beispiele von Lossfunktionen. Recherchiert u.a. hier: https://keras.io/api/losses/.

Wahl ist abhängig von Task, Aktivierungsfunktion und Charakteristik der Daten.

Cross-Entropy Loss	→ Gut geeignet für Multilabel-Klassifikation → Fördert Optimierung des Modells und damit die Ausgabe klarer Entscheidungen zwischen den verschiedenen Klassen → Anfällig für Overfitting bei unausgewogenen Daten
Binary Cross-Entropy Loss	→ Gut geeignet für binäre Klassifikation → Kann bei unausgewogenen Daten instabil werden

→ Optimierungsfunktion

Es folgen drei Beispiele von Optimierungsfunktionen. Recherchiert u.a. hier: https://keras.io/api/optimizers/ .

Wahl ist abhängig von Task, Charakteristik der Daten und Hyperparameter-Einstellung.

Stochastic Gradient Descent (SGD)	→ Gut für große Datensätze → Einfach verständlich → Gefahr von Divergenz; Learning Rate muss gut angepasst werden
Adam (Adaptive Moment Estimation)	→ Adaptive Anpassung der Learning Rate für jeden Learning Step → Braucht mehr Ressourcen als SGD

→ <u>Netzarchitektur (Feedforward-Netze)</u>

Tiefe Netze	→ Fähig komplexe Muster in den Daten zu erkennen → Neigen besonders bei kleinen Datensets zu Overfitting → Brauchen mehr Ressourcen und Daten für das Training
Flache Netze	→ Robuste, leichter interpretierbare Ergebnisse → Weniger Overfitting → Begrenzte Erfassung komplexer Datenstrukturen

<u>Troubleshooting bei Overfitting:</u>

→ L2 Regularization

ⓘ	→ Stärkere Regularisierung und damit stärkere Veränderung des Loss und der Hypothesenfunktion → Reduziert Overfitting → Zu hoher Wert kann zu Informationsverlust und damit zu Underfitting führen
ⓘ	→ Schwächere Regularisierung → Gefahr von Overfitting → Ermöglicht genauere Anpassung an Trainingsdaten
Empfehlung	→ Experimentieren, um Balance zwischen Under- und Overfitting zu schaffen

→ Drop Out

ⓘ	→ Höhere Wahrscheinlichkeit, dass Neuronen während des Trainings deaktiviert werden → Reduziert Overfitting → Zu hoher Wert kann zu Informationsverlust und damit zu Underfitting führen

ⓘ	→ Geringerer Zufälligkeitswert; weniger Neuronen werden deaktiviert → Gefahr von Overfitting → Ermöglicht genauere Anpassung an Trainingsdaten
Empfehlung	→ Experimentieren, um Balance zwischen Under- und Overfitting zu schaffen

→ Honorable Mention: Data Augmentation (wenn möglich)

- Mehr Kontext einbeziehen
  - Besseres Verständnis von Zusammenhängen und Beziehungen zwischen Wörtern und erhöhte Berücksichtigung von Kontextinformationen
  - Genauere Repräsentation komplexer semantischer und syntaktischer Strukturen
- Mehr Daten
  - Bessere Generalisierung; weniger Overfitting
  - Robustheit gegenüber Rauschen und Unregelmäßigkeiten in den Trainingsdaten; höhere Variation
- Balancierung der Labels
  - Gleichgewicht verhindert, dass sich das Modell auf die häufigere Klasse konzentriert und die seltenere Klasse vernachlässigt
  - Führt zu generaueren Vorhersagen aller Klassen
  - Einfachere Interpretation der Ergebnisse