Weitere Untersuchungen hinsichtlich der Anwendung von KNN für Solvency 2

Tom Schelthoff

30.11.2018

Inhaltsverzeichnis

- Deep Learning
- Seed-Stabilität
- Regularisierung
  - Early Stopping
  - Dropout
  - Batch Normalization

DEEP LEARNING

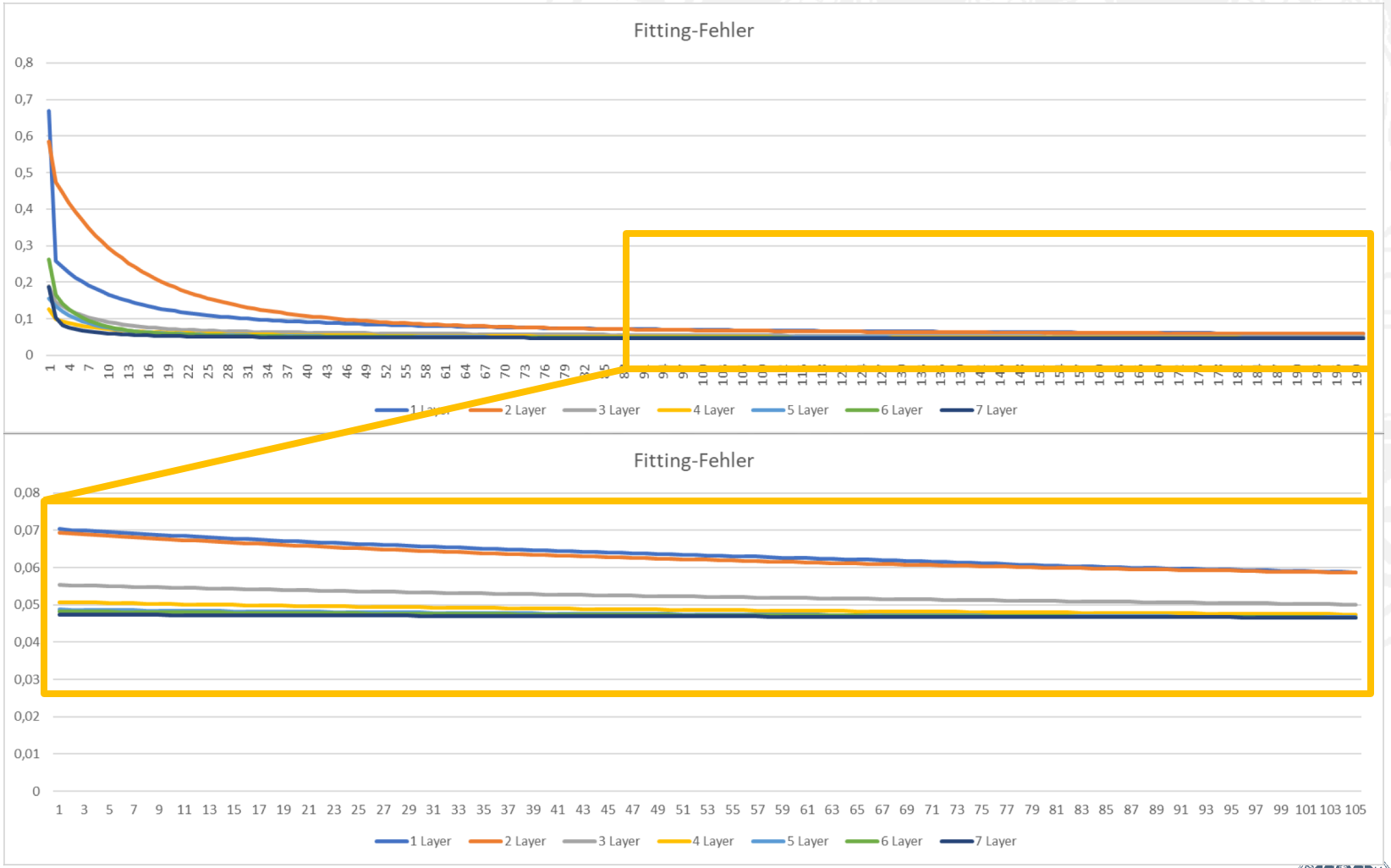
Deep Learning

- Universal Approximation Theorem (1988):
  - Jede (glatte) Funktion kann durch ein KNN mit einem Hidden Layer approximiert werden
- Probleme:
  - Anzahl Neuronen in diesem Layer unbekannt und ggf. groß
  - Trainingsgeschwindigkeit
- Deep Learning: Mehr als ein Hidden Layer

Deep Learning und Regression

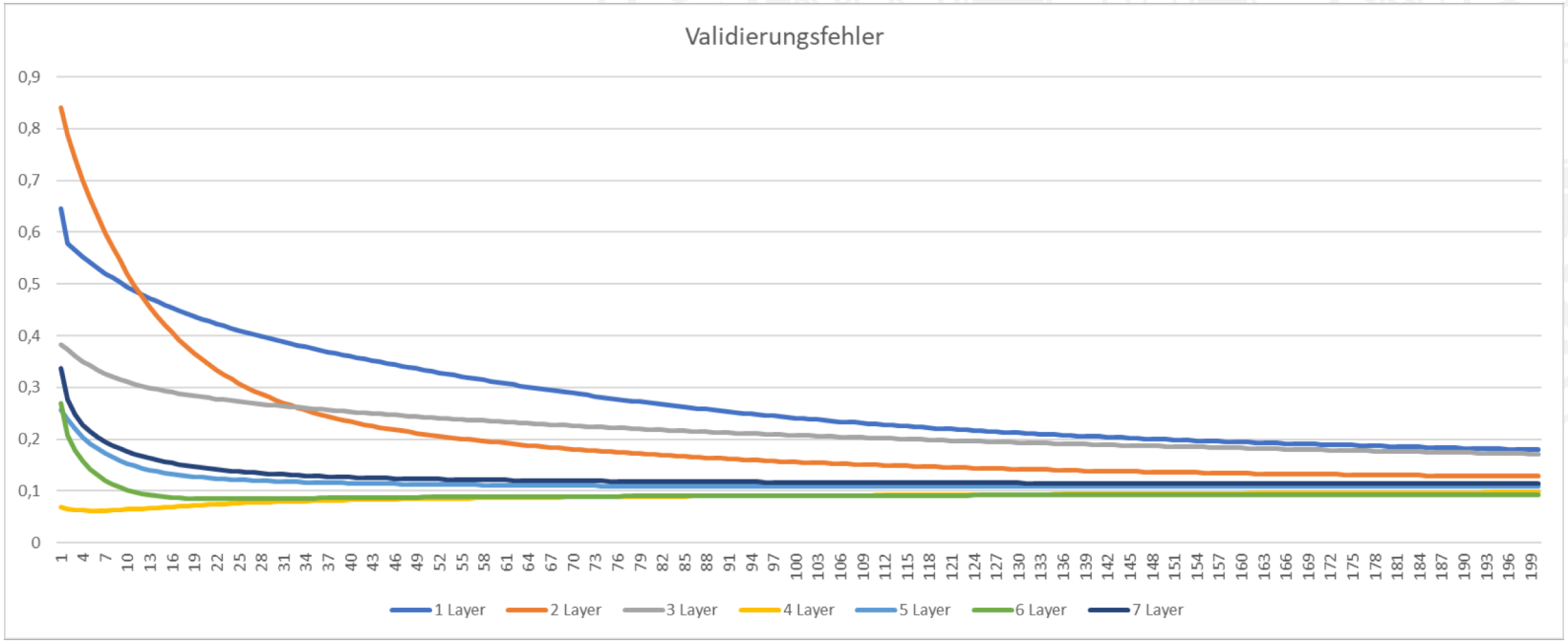
- **Analogie in der Regression:**
  - **Terme mit höherer Ordnung und Korrelationsterme**
  - **Jeder Layer kann als um 1 erhöhte Grade gesehen werden**
- **Zahl der Parameter**
  - **Regression vom Maximalgrad N bei m Inputs:**
    - $1 + m + m^2 + m^3 + \cdots + m^N = \sum_{j=0}^{N} m^j$
  - **Neural Net mit N hidden Layern mit jeweils m Neuronen:**
    - $\boldsymbol{N * m^2 + m}$

Deep Learning - Fitting



Deep Learning - Validierung

- Höhere Terme reduzieren den Fehler auf den Fittingpunkten
  - Aber: Muss nicht für tatsächliche Punkte gelten



Deep Learning

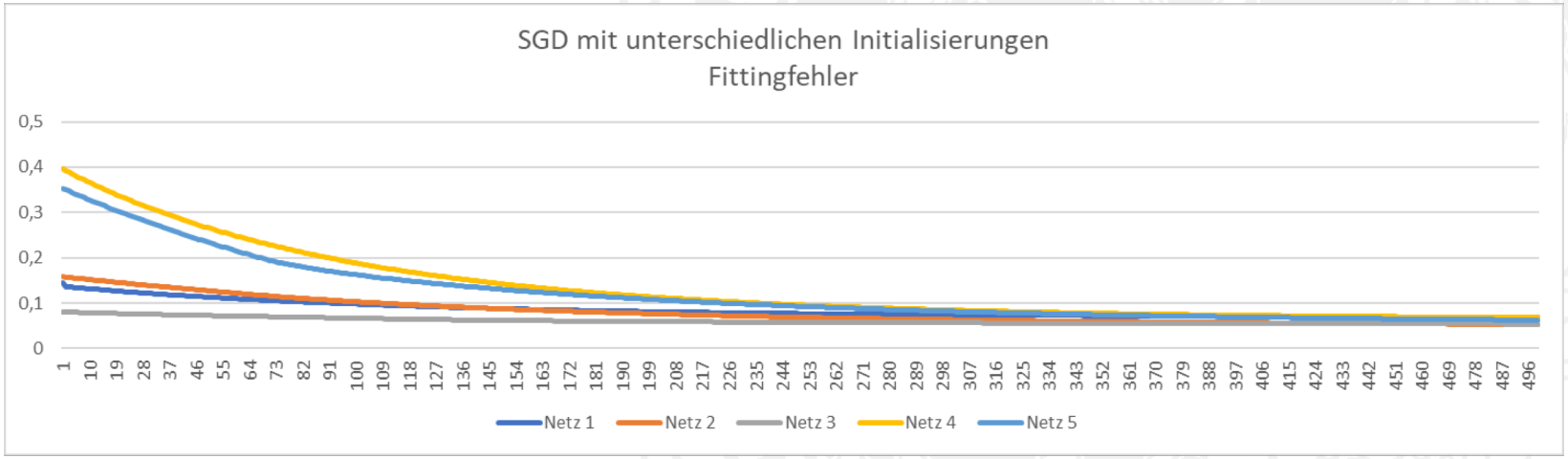
- Warum performen größere Netze teilweise schlechter?
  - Overfitting
  - Langsames Training in frühen Layern
  - Dimensionalität
  - Seed-Stabilität

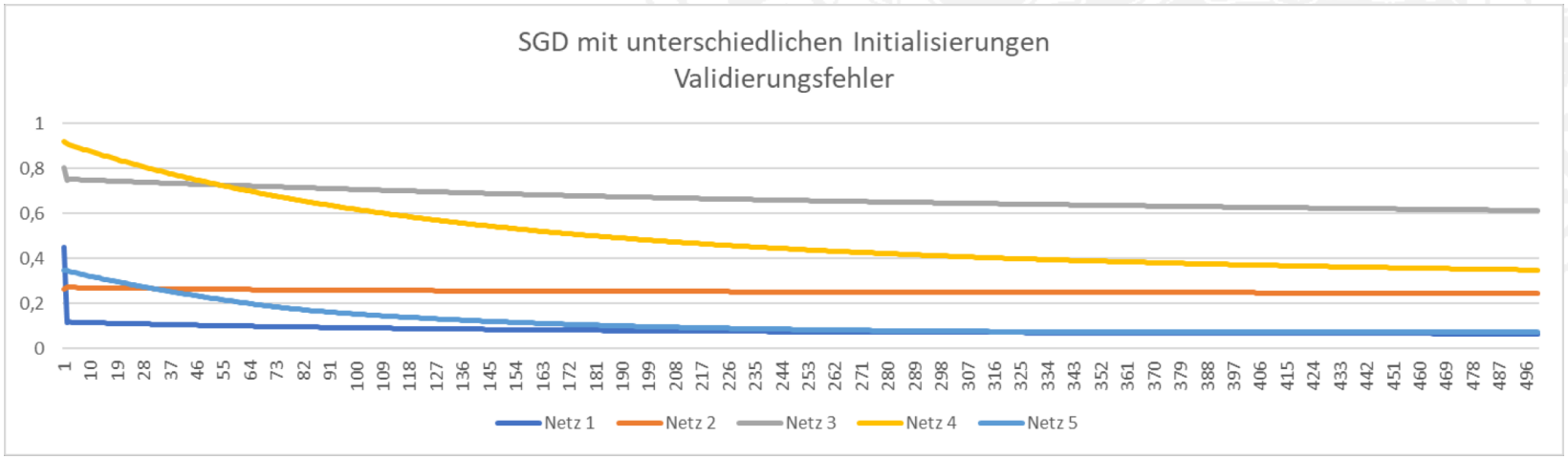
SEED-STABILITÄT

Seed-Stabilität

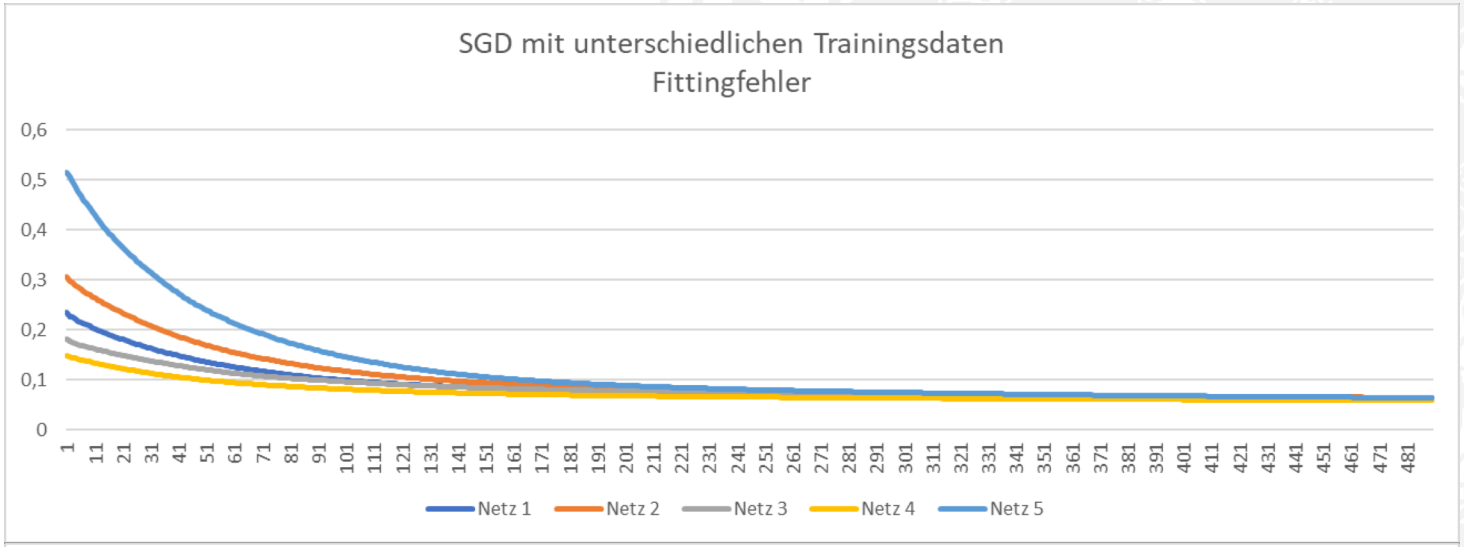
- Verhalten bei unterschiedlichen Initialisierungen
  - Zufällige Start-Gewichte
  - Validierungsset
- Verhalten bei unterschiedlichen Fitting-Datensätzen
  - Insbesondere bei fehlerbehafteten LSMC-Daten
  - Wichtig für öfter trainierte Modelle

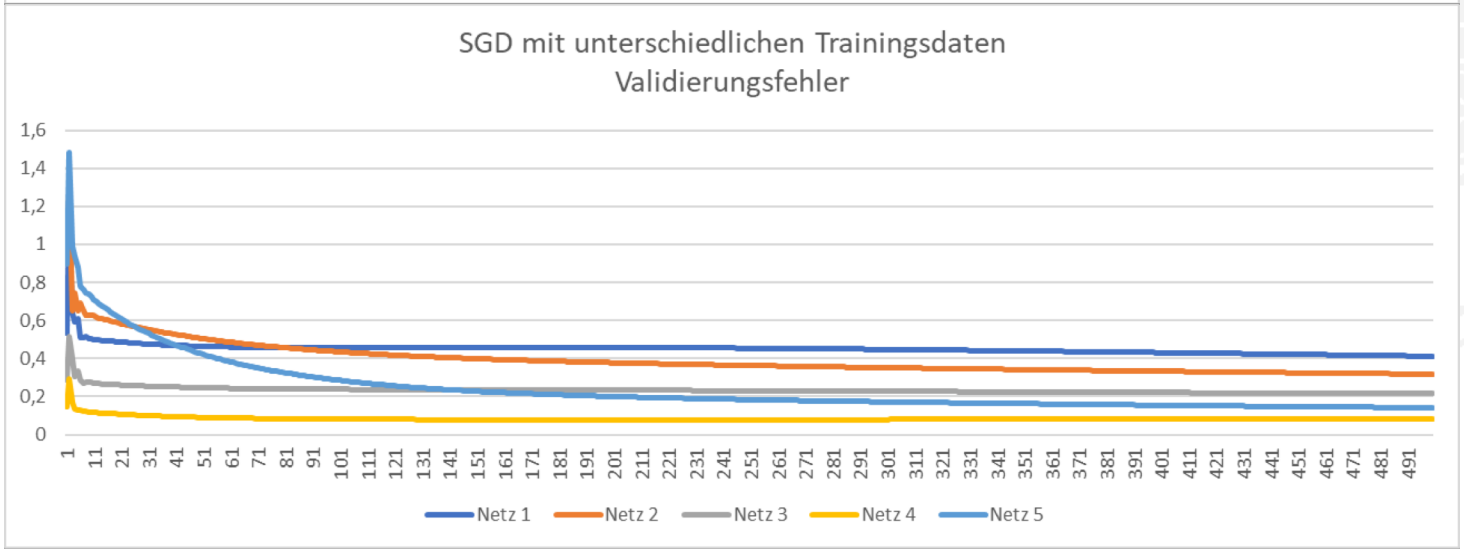
Seed-Stabilität





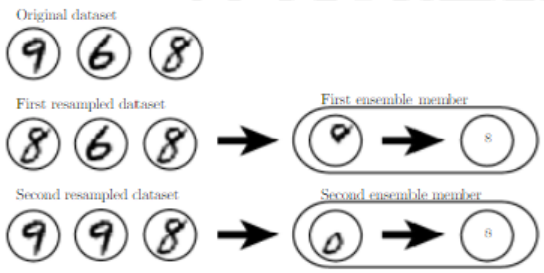
Seed-Stabilität





Bootstrap Aggregation / Ensemble

- Trainiere mehrere Modelle unabhängig voneinander und mittele sie am Ende
- Idee: Unabhängig trainierte Modelle machen unterschiedliche Fehler, die sich im Mittel herauskürzen
- Erzeuge „unabhängige“ Trainingsdaten durch Bootstrapping: Ziehen mit Zurücklegen aus ursprünglichem Datensatz
- Sind die Bootstrap-Datensätze gleich groß wie der ursprüngliche, so enthält jeder von ihnen im Schnitt nur 2/3 der Datenpunkte
- Beispiel: 8-Detektor:



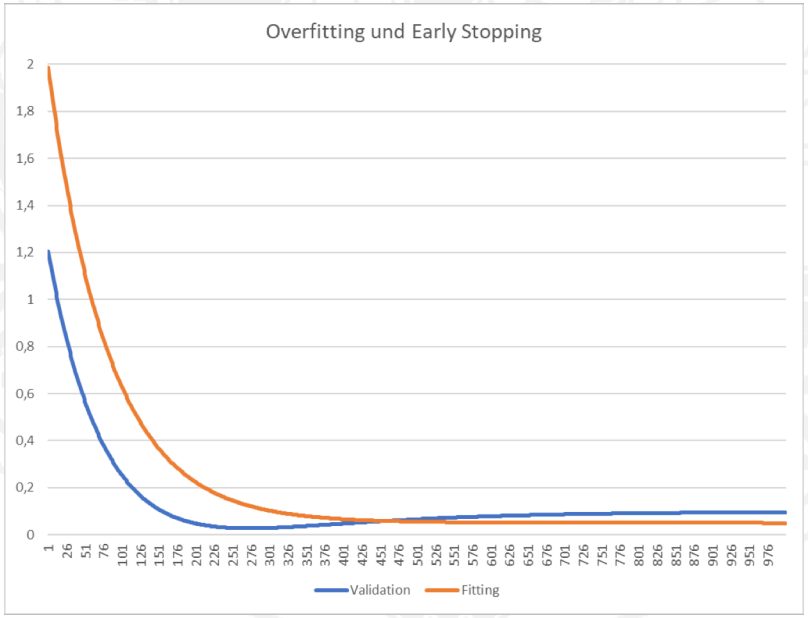
REGULARISIERUNG

Regularisierung

- Reduktion des Test-Fehlers
- Oft auf Kosten des Trainingsfehlers
- Basierend auf vorhandenem Wissen oder der Präferenz einer simplen Lösung
- Varianz reduzieren gegen eine kleinere Erhöhung des Bias

Early Stopping

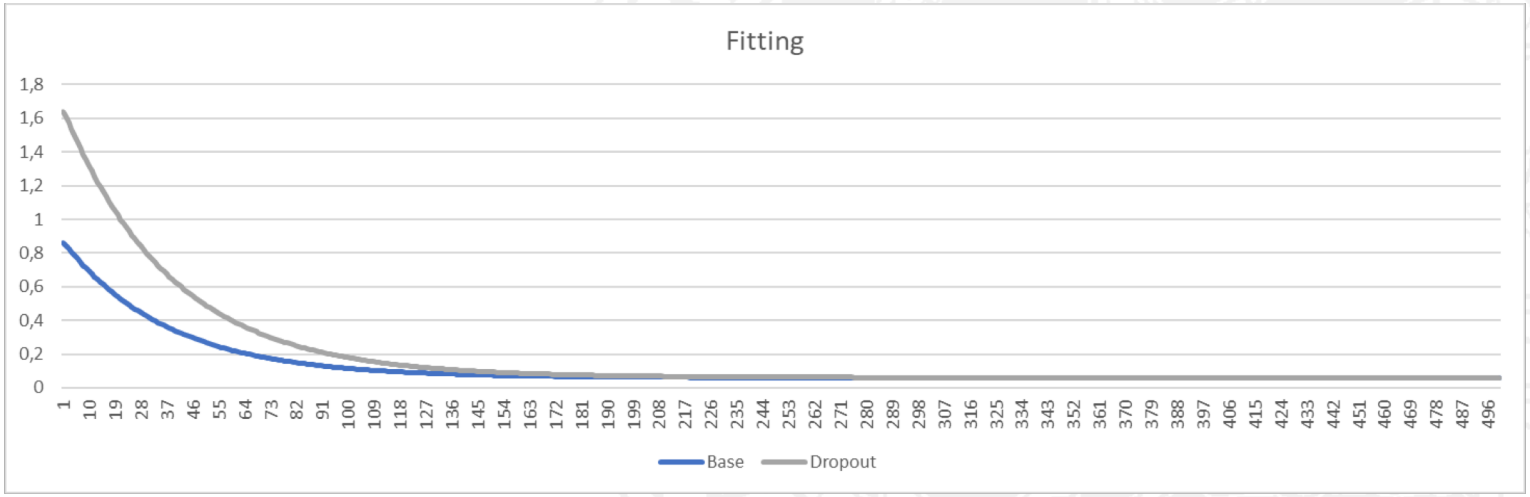
- Zu langes Trainieren führt zu Overfitting
- Trainingsfehler fällt, aber Validierungsfehler steigt wieder
- Lösung: Speichern der Modellparameter mit dem geringsten Validierungsfehler und Ausgabe deren. Abbruch wenn in einer vorgegebenen Anzahl Schritte keine Verbesserung erreicht wurde
- Zusätzliche Kosten gering, Validierung kann parallel durchgeführt werden
- Aber: Teil des Datensatzes nicht genutzt

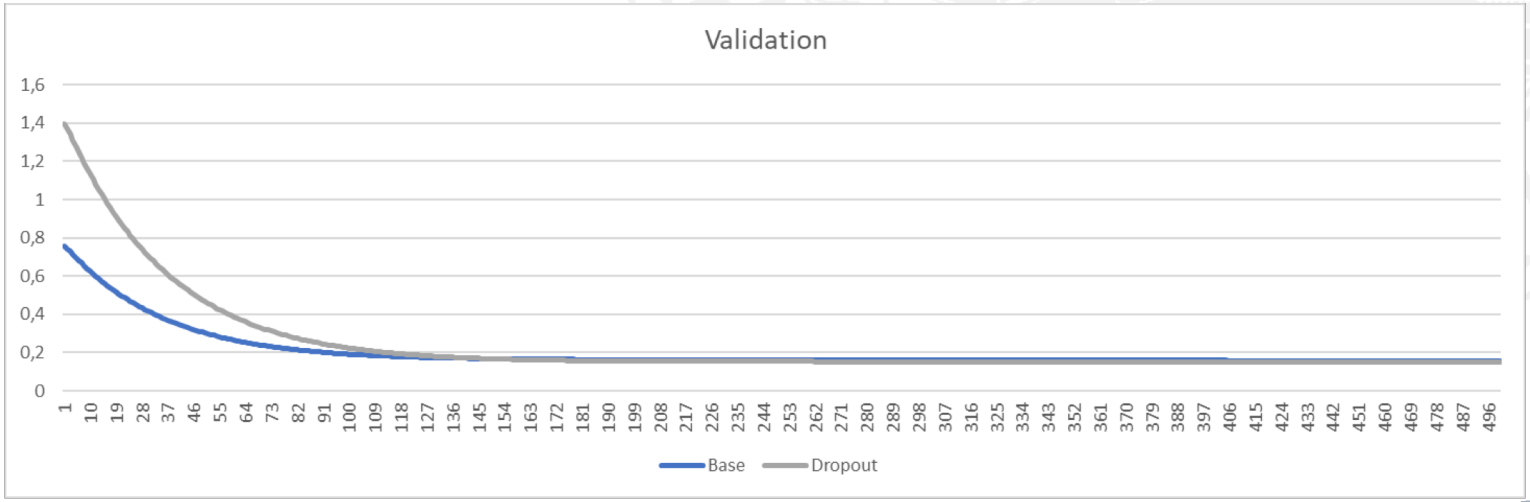


Dropout

- Idee: Verhindern, dass einzelne Neuronen zu große Wichtigkeit erhalten
- Ansatz: Deaktivieren zufälliger Neuronen in jeder Iteration
- Effekt: Netzwerkfluss wird über andere Gewichte umgeleitet

Dropout für LSMC





BATCH NORMALIZATION

Batch Processing

- Anstelle des Berechnens des Gradienten für alle Input Daten oder nur einzelne, kann man die Daten in Batches der Größe m aufteilen
- Benutze dann $\frac{1}{m}\sum_{i=1}^{m}\frac{\partial l(x_i;\theta)}{\partial \theta}$ zum Update der Parameter θ
- Vorteile: Bessere Approximation des Gradienten als einzelne Beispiele und wenn geeignet Parallelisiert sogar schneller

Internal Covariate Shift

- Durch Trainingsupdates der Gewichte gibt es eine Änderung der Verteilung der Outputs der (inneren) Neuronen
- Zwang zur Nutzung niedriger Lernraten, um Divergenz zu vermeiden, wegen Fortpflanzung der Gewichtsänderungen
- Schwierigkeiten bei Nutzung von z.B. der Sigmoidfunktion als Aktivierungsfunktion, aufgrund von Saturierung

Normalization

- LeCun, Bottou, Orr und Muller, 1998: Schnelleres Training auf normalisierten und dekorrelierten Daten
- Idee: Solche Inputs für jeden Layer, nicht nur den Input-Layer
- Zur Vereinfachung normalisieren wir jede Inputdimension einzeln: Sei $x = (x^1, \dots, x^d)$ der Input eines Layers. Dann normalisieren wir jede Dimension $\hat{x}^k = \frac{x^k - E[x^k]}{\sqrt{Var(x^k) + \varepsilon}}$, wobei Erwartungswert und Varianz empirisch aus dem aktuellen Batch berechnet werden
- Um nicht an Allgemeinheit zu verlieren, müssen wir danach neu skalieren $Y^k = \gamma^k \hat{x}^k + \beta^k$
- Die Parameter γ und β werden dann während des Trainings gelernt
- Wir bezeichnen die gesamte Transformation durch $BN_{\gamma,\beta}: x_{1,\dots,m} \to Y_{1,\dots,m}$

Batch Normalization

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

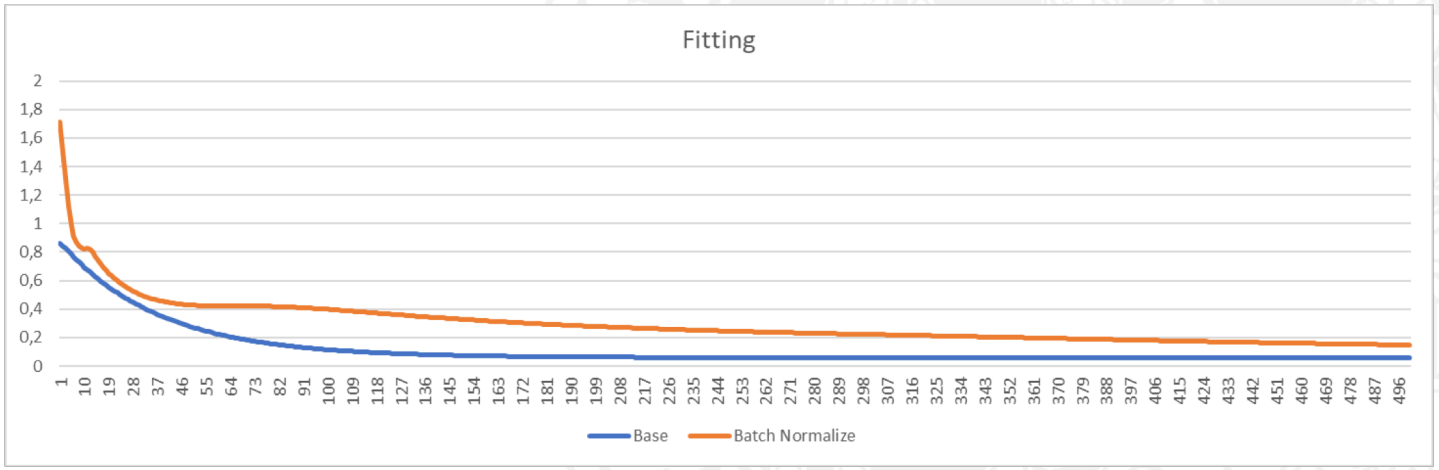
1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

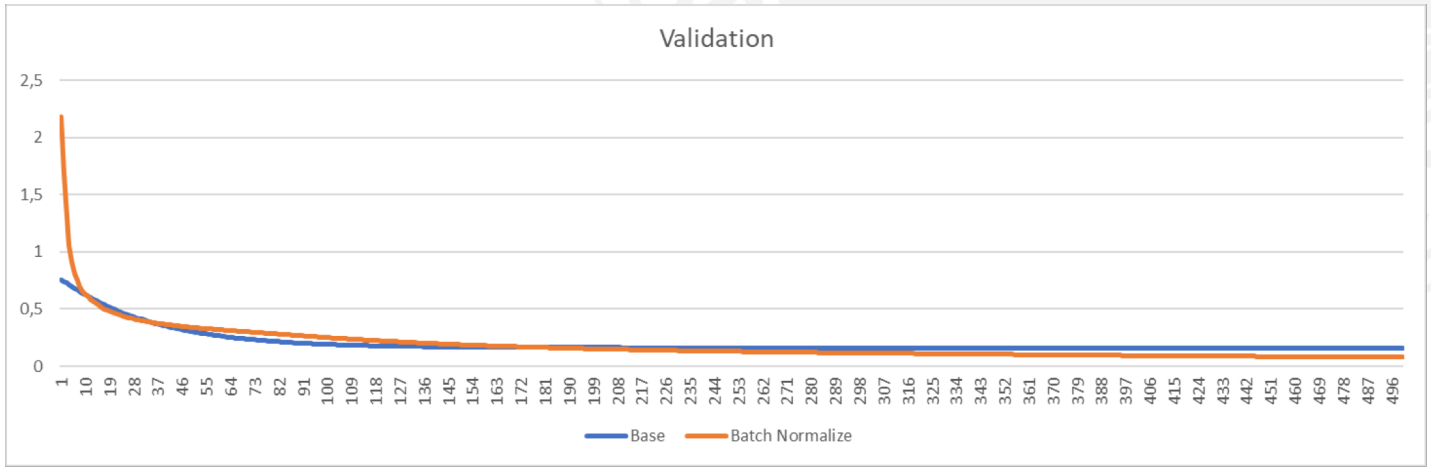
$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Batch Normalization für LSMC





VIELEN DANK FÜR DIE AUFMERKSAMKEIT