Seminar Machine Learning

Trainieren eines Neuronalen Netzwerkes

Dozent: Zoran Nikolić

David Noben, Marcel Aach

28. Juni 2019

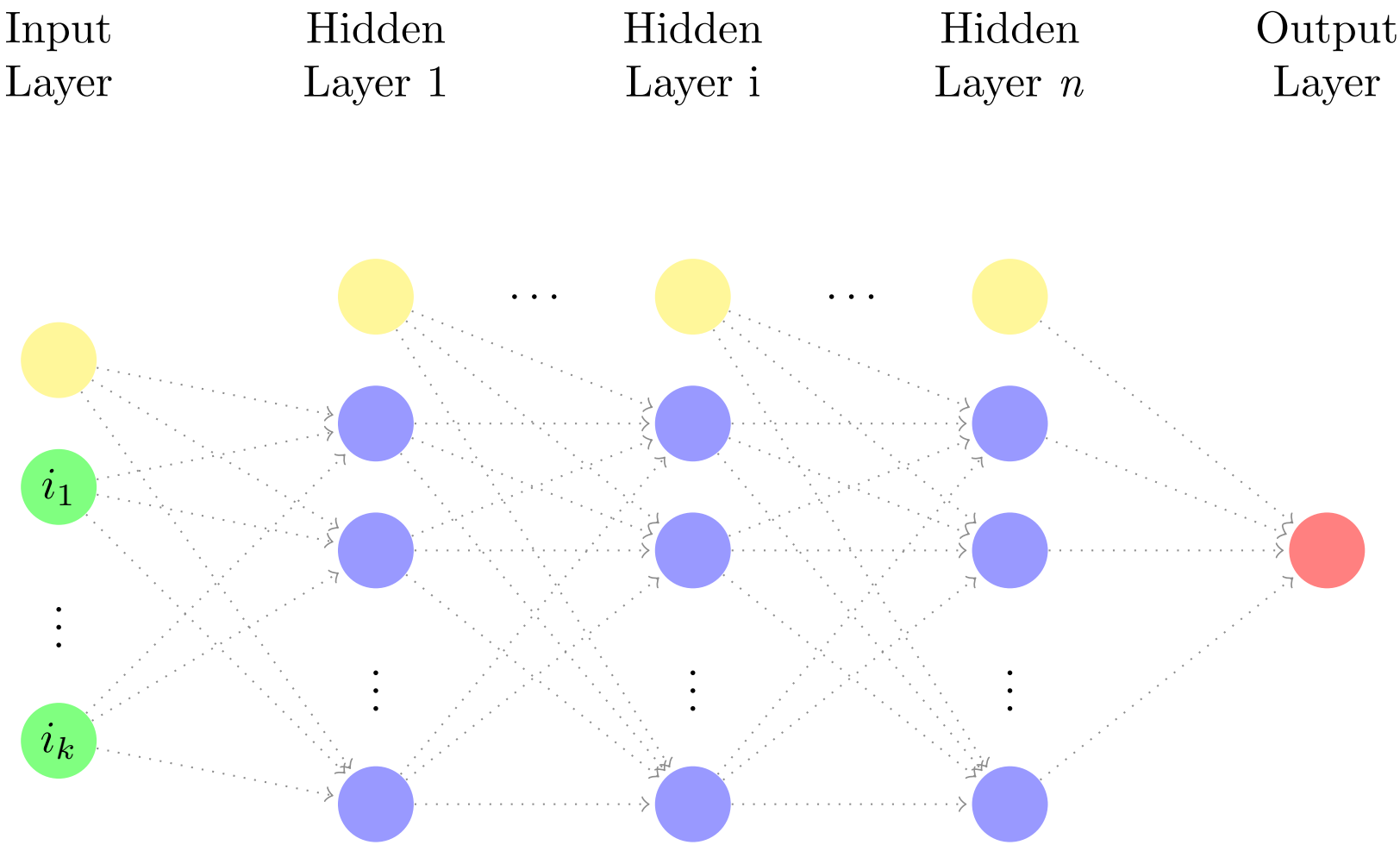
Vorgehen

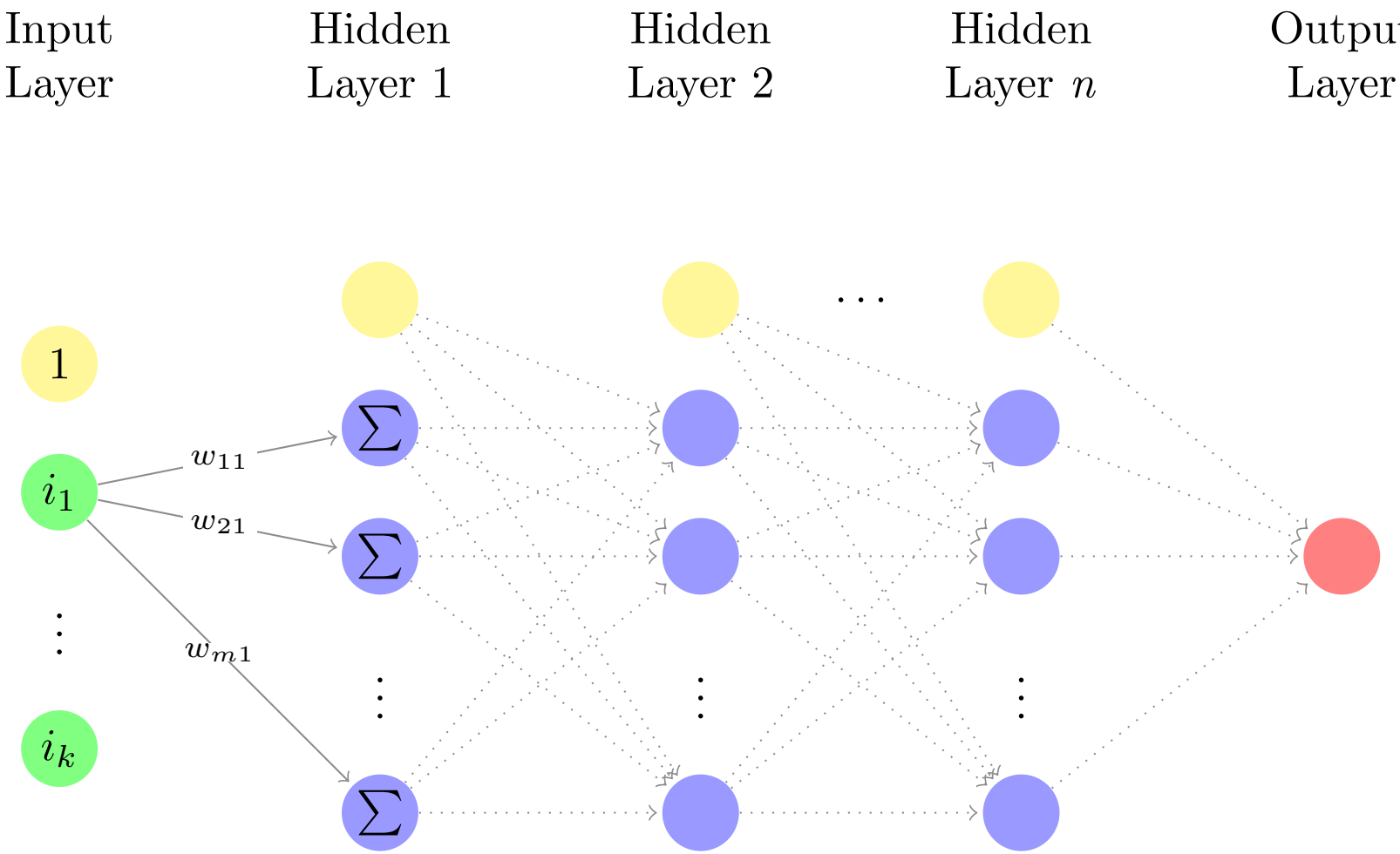
Ergebnisse

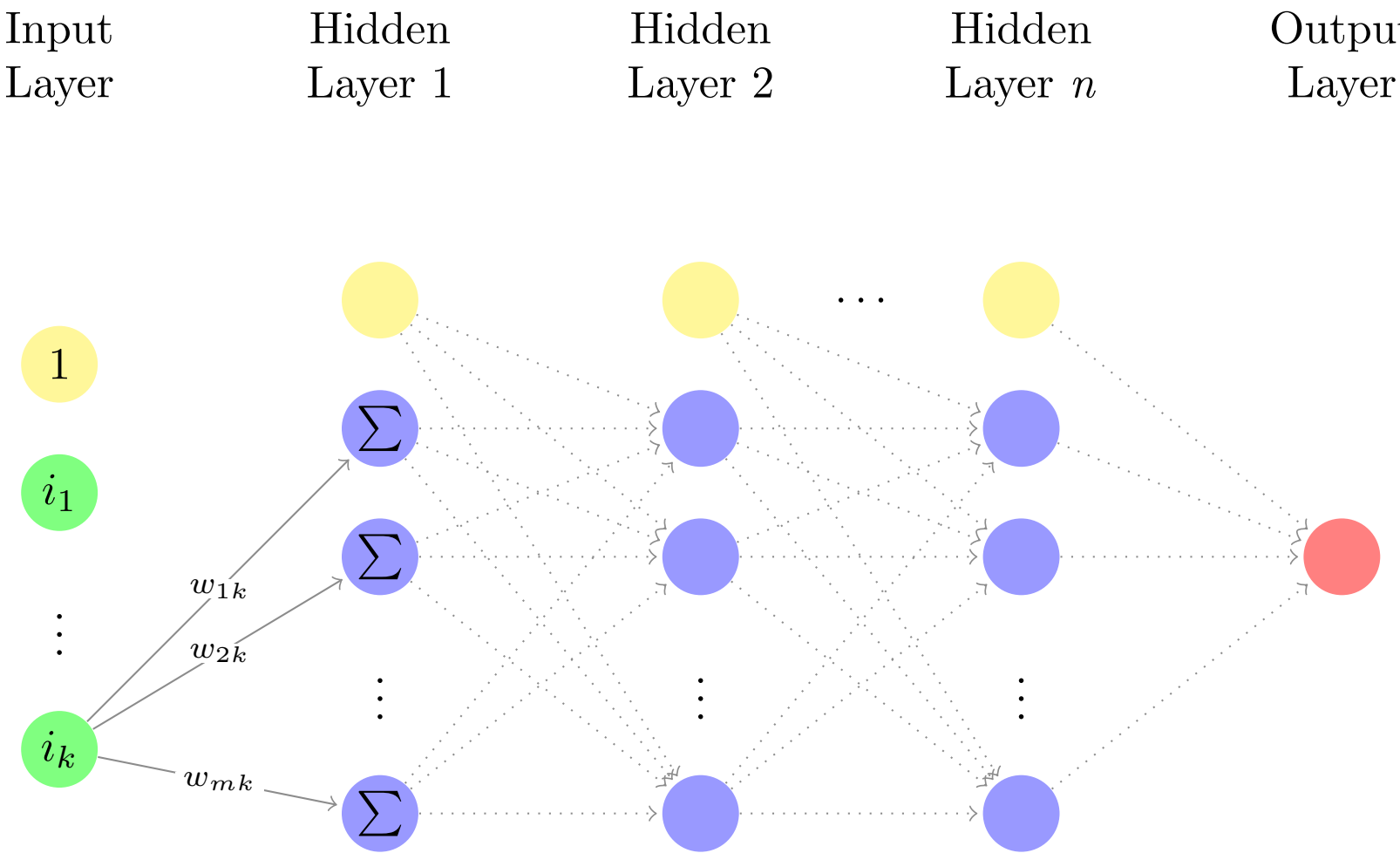
Ensemble

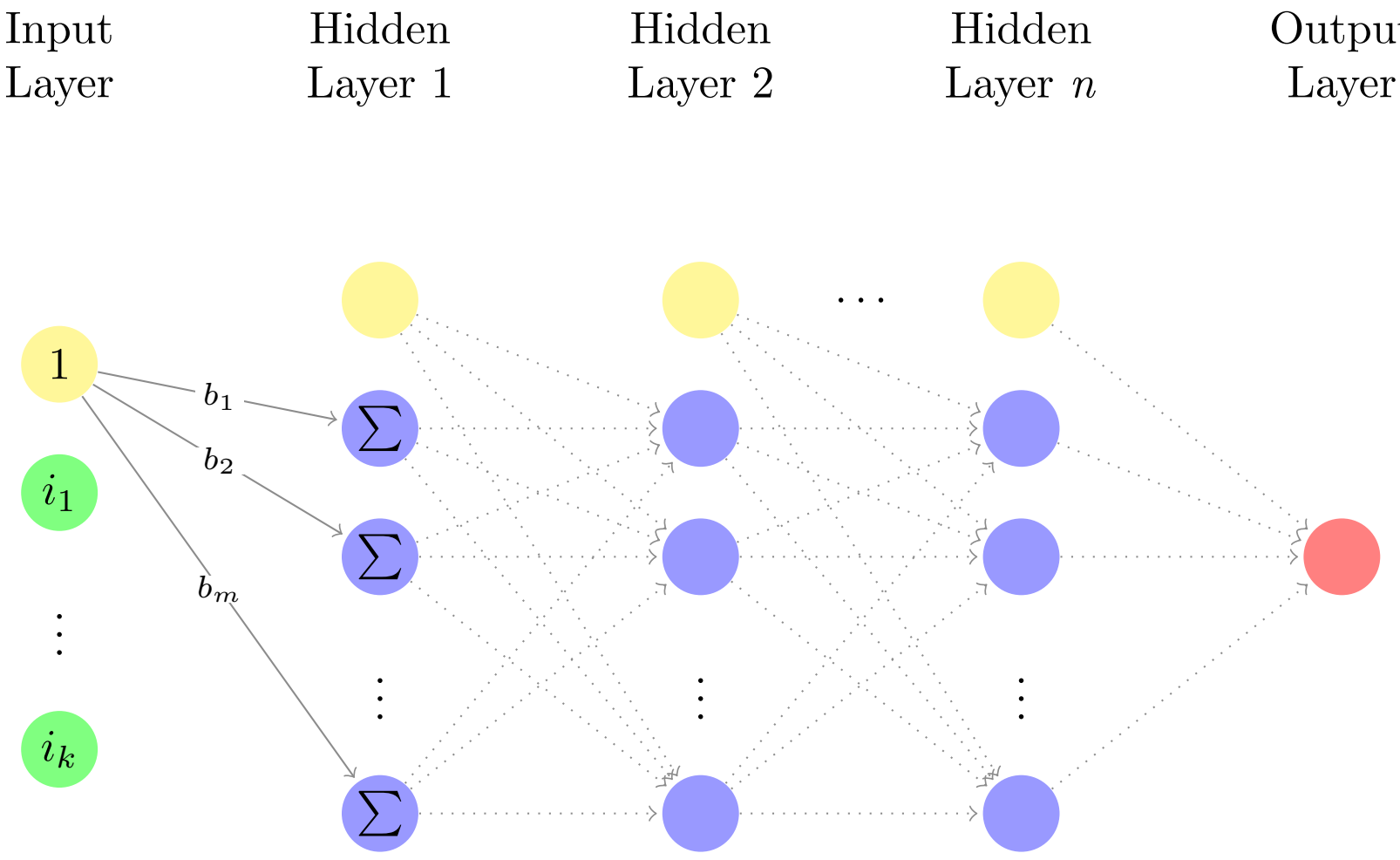
Performance

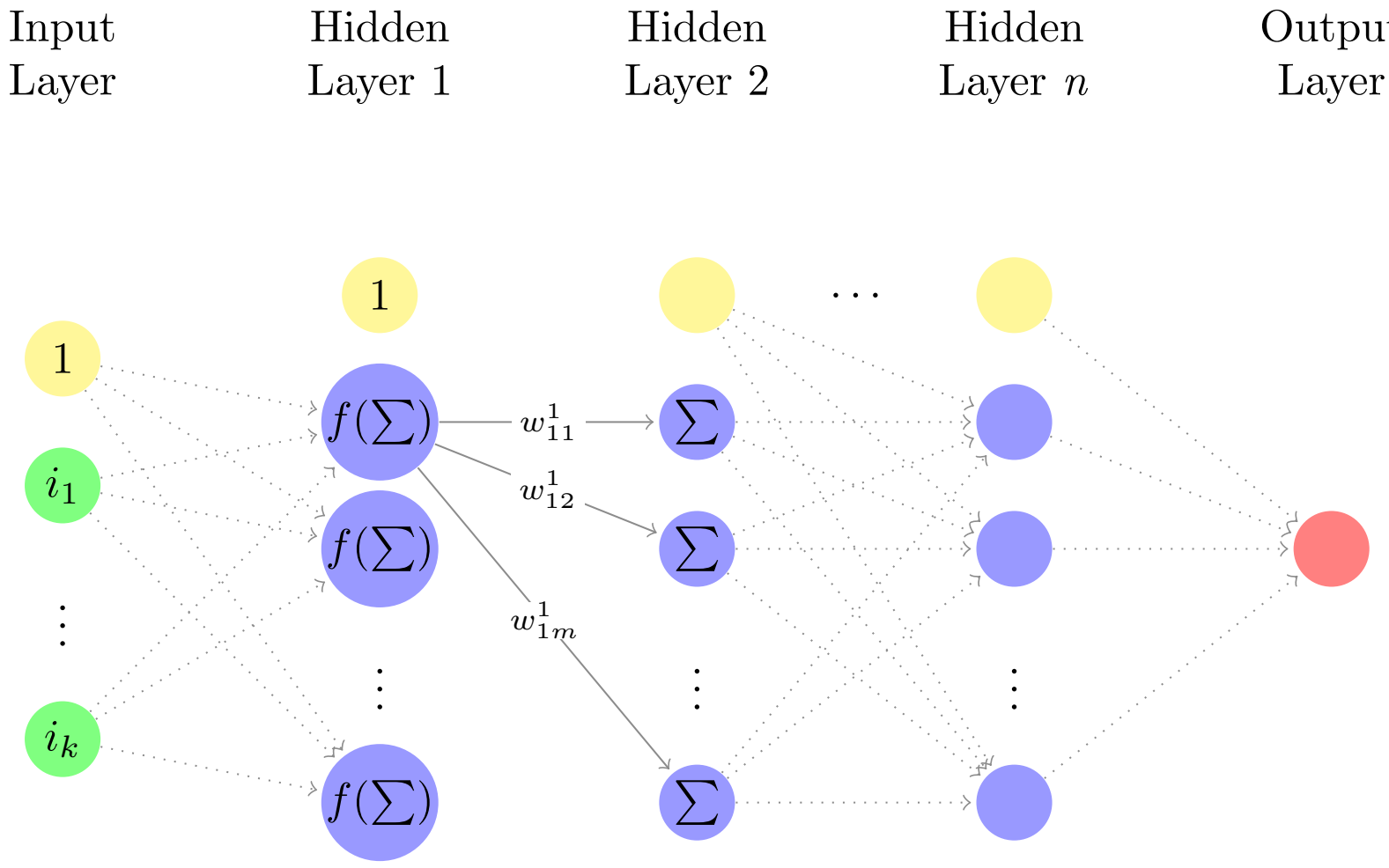
Literatur

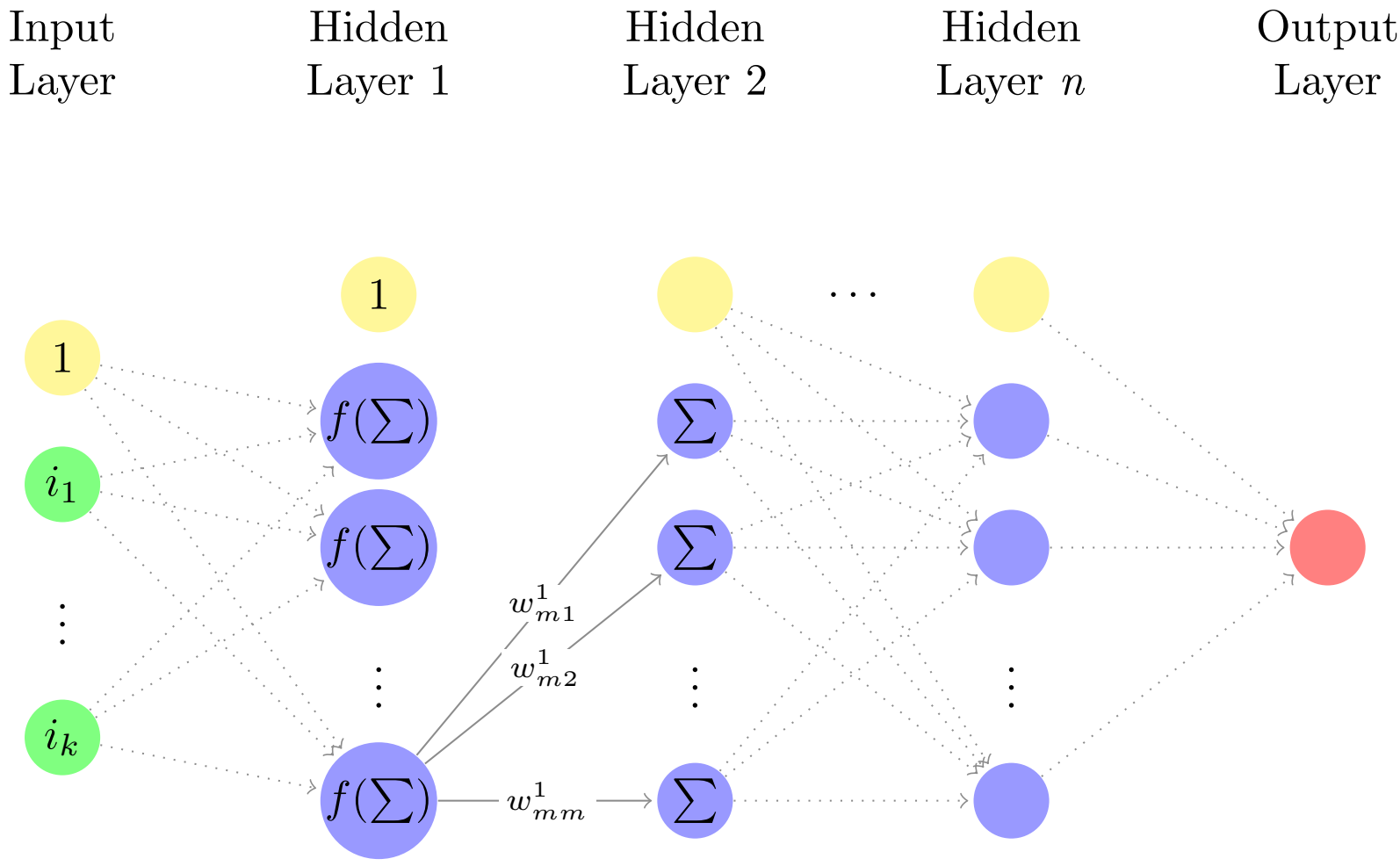

Input Layer
Hidden Layer 1
Hidden Layer i
Hidden Layer n
Output Layer
i_1
i_k

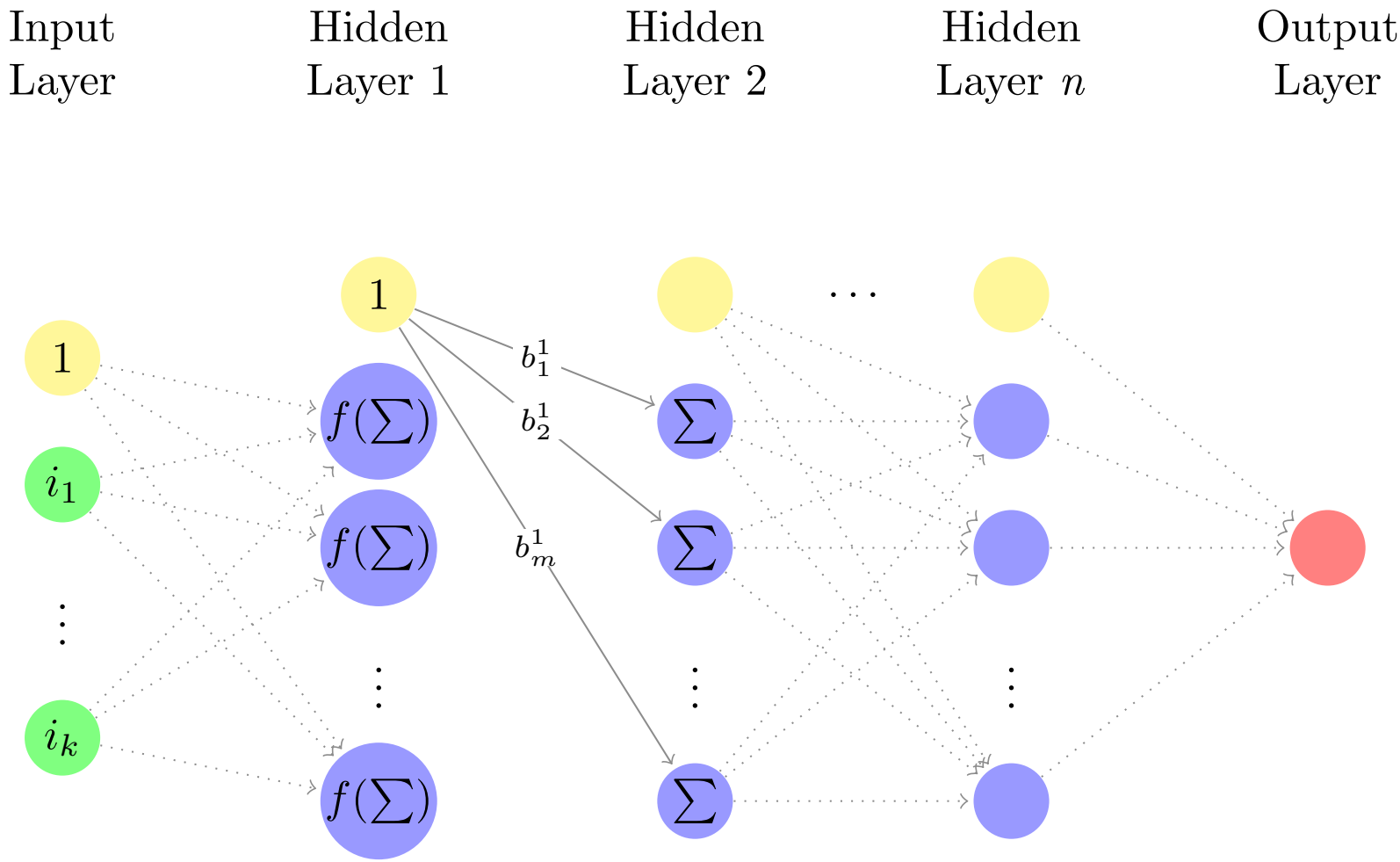

Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer n
Output Layer
1
i_1
i_k
w_{11}
w_{21}
w_{m1}
$\sum$
$\sum$
$\sum$
. . .

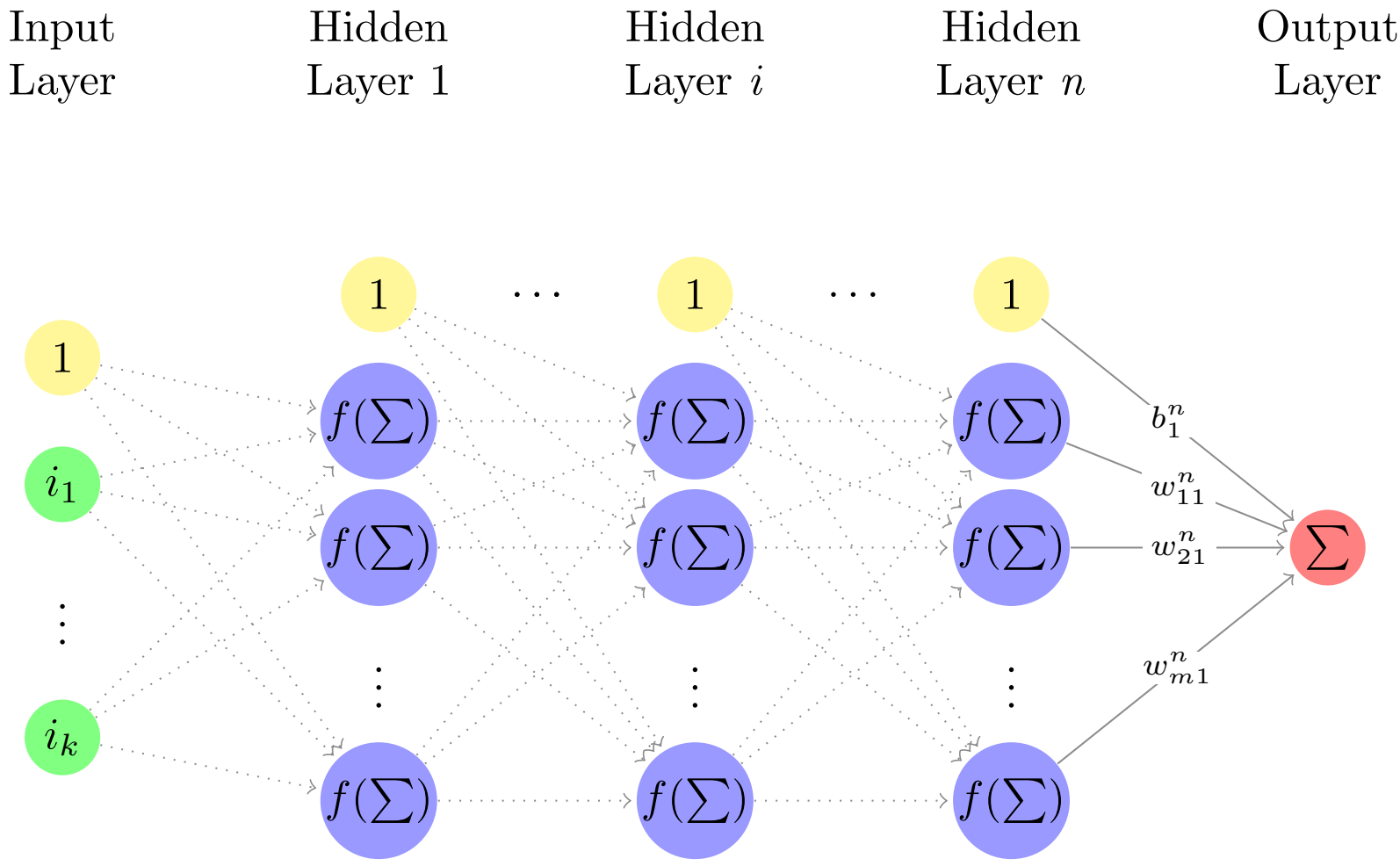

Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer n
Output Layer
1
i_1
i_k
w_{1k}
w_{2k}
w_{mk}
$\sum$
$\sum$
$\sum$
...


Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer n
Output Layer
1
i_1
i_k
b_1
b_2
b_m
$\sum$
$\sum$
$\sum$
...


Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer n
Output Layer
1
i_1
i_k
1
$f(\sum)$
$f(\sum)$
$f(\sum)$
w_{11}^1
w_{12}^1
w_{1m}^1
$\sum$
$\sum$
$\sum$
...

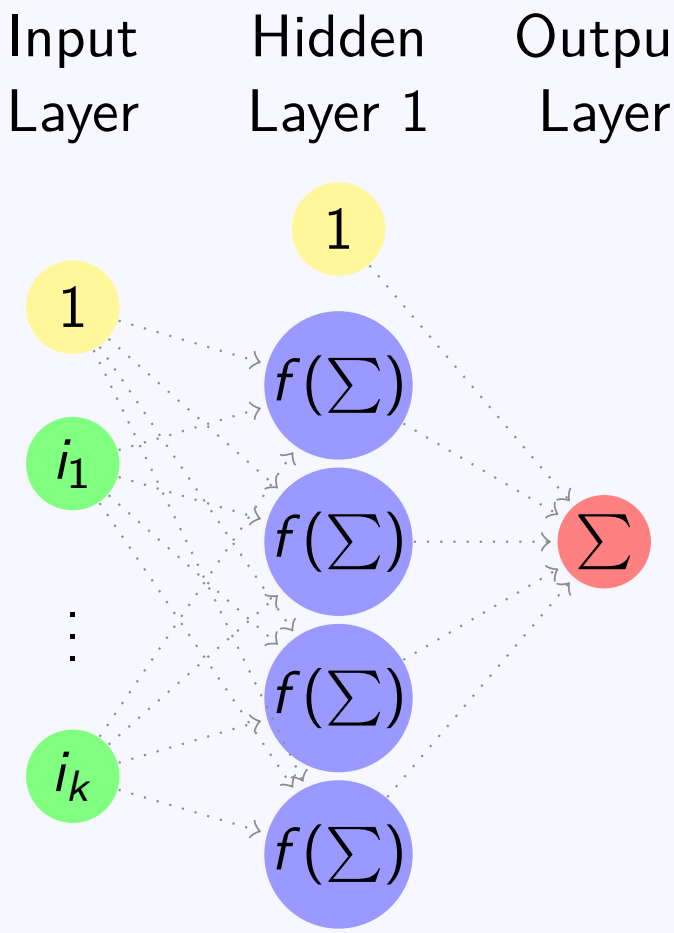

Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer n
Output Layer
1
i_1
i_k
1
$f(\sum)$
$f(\sum)$
$f(\sum)$
$\sum$
$\sum$
$\sum$
w^1_{m1}
w^1_{m2}
w^1_{mm}


Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer n
Output Layer
1
i_1
i_k
1
$f(\sum)$
$f(\sum)$
$f(\sum)$
b_1^1
b_2^1
b_m^1
$\sum$
$\sum$
$\sum$
...


Input Layer
Hidden Layer 1
Hidden Layer i
Hidden Layer n
Output Layer
1
i_1
i_k
$f(\sum)$
b_1^n
w_{11}^n
w_{21}^n
w_{m1}^n
$\sum$

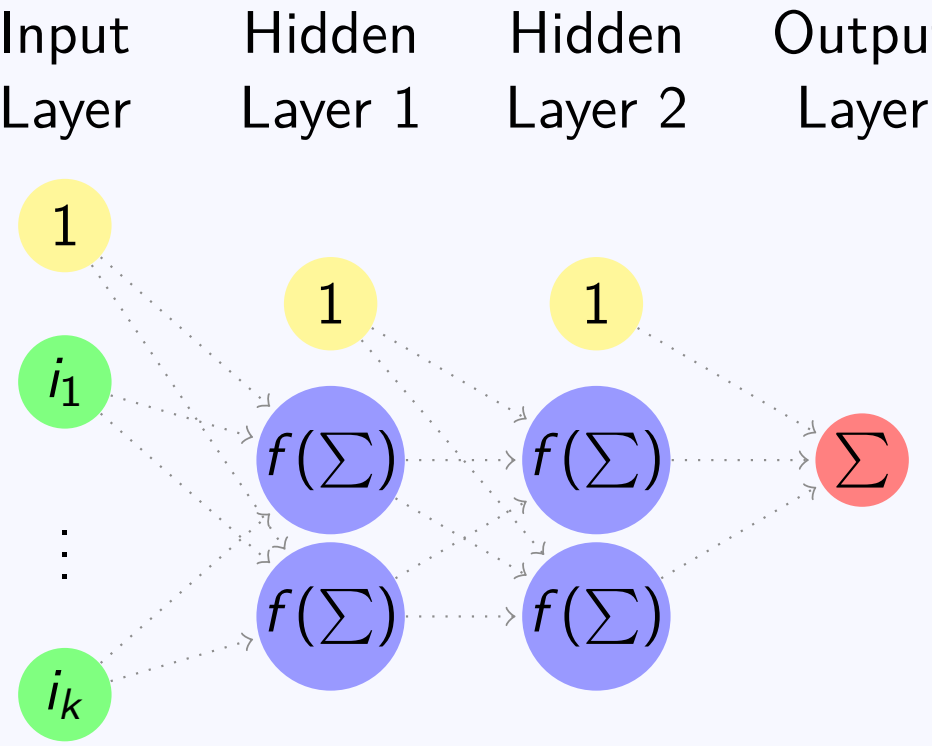
Vorgehen

Feedforward Netzwerk, Anzahl an Neuronen pro Hidden Layer 4 und 1 Hidden Layer:



KNN<4,1>

Feedforward Netzwerk, Anzahl an Neuronen pro Hidden Layer 2 und 2 Hidden Layer:



KNN<2,2>

Stochastisches Gradientenverfahren (SGD)

Sei n die Anzahl der Trainingsdaten. Mit $w \in \mathbb{R}^m$ bezeichnen wir den Vektor aller zu optimierenden Parameter, um die Fehlerfunktion $R(w) = \sum_{i=1}^{n} R_i(w)$ zu minimieren.

Require: Lernrate $\eta > 0$, Abbruchkriterium.
  Initialisiere w
  while Abbruchkriterium nicht erfüllt **do**
    Bilde zufällige Reihenfolge der Trainingsdaten $\{1, \ldots, n\}$
    for i=1 to n **do**
      $w^{(t)} = w^{(t)} - \eta \nabla_w R_i(w^{(t)})$
    end for
    $w^{(t+1)} = w^{(t)}$
    $t = t + 1$
  end while
  return w

Warum kann das SGD-Verfahren von Vorteil sein?

(i) Für einzelne Updates wird nicht die gesamte Datenmenge benötigt: Updates sind günstiger zu berechnen

(ii) Die erhöhte Update-Frequenz kann zu einer schnelleren Konvergenz führen

(iii) Insbesondere dann, wenn viele Trainingsdaten redundant sind

(iv) Durch die Updates mit variierenden Gradienten, kann die Konvergenz an (schlechte) lokale Minima übersprungen werden

Mini-Batch SGD

In der Praxis wird das Verfahren meist als **Mini-Batch** Variante implementiert. Dazu werden in jeder Iteration zufällig Teilmengen $\mathcal{I}_i \subset \{1, \ldots, n\}$ der Größe $k < n$ (**Batch-Size**) mit

$$\dot{\bigcup_{i=1}^{\lceil n/k \rceil}} \mathcal{I}_i = \{1, \ldots, n\}$$

gebildet. Statt des Gradienten einer einzelen Trainingsdatei, werden die Updates der Parameter nun mit

$$\nabla_w R_{\mathcal{I}_i}(w) = \sum_{j \in \mathcal{I}_i} \nabla_w R_j(w)$$

berechnet. Einen gesamten Update-Durchlauf aller Mengen $\mathcal{I}_i, i = 1, \ldots, \lceil n/k \rceil$ nennt man **Epoche**.

Warum Mini-Batch SGD?

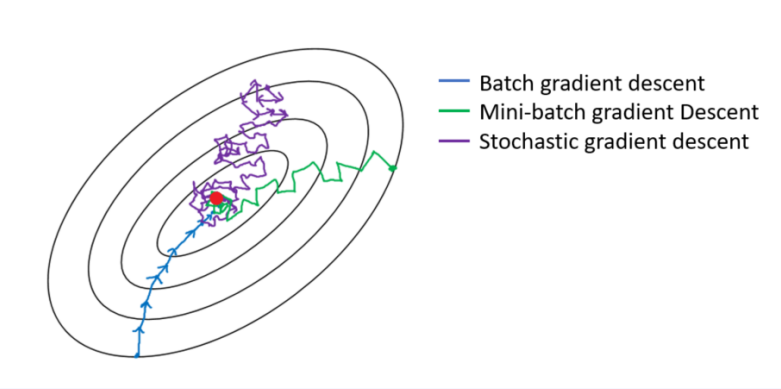


Abbildung: Vergleich der Varianten, entnommen aus [Dab17]

(i) Parallele Rechenarchitektur eines Computers wird besser ausgenutzt. Insbesondere Vektor-Additionen und Vektor-Multiplikationen (z.B. SIMD-Einheiten)

(ii) Weniger stark variierende Gradienten führen zu einer stabileren Konvergenz

(iii) Insgesamt guter Tradeoff zwischen stochastischem und normalem Gradientenverfahren

Allgemeine Probleme mit Gradientenverfahren (Wiederholung)

Problem 1: Wie soll die Lernrate η gewählt werden?

(i) Zu kleines η führt zur langsamen Konvergenz und hohen Berechnungskosten

(ii) Zu großes η lässt den Algorithmus divergieren

(iii) Gewichte $w_i \in w$ können stark variieren. Eine Lernrate für alle Parameter kann kritisch sein

Allgemeine Probleme mit Gradientenverfahren (Wiederholung)

Problem 2: Abhängigkeit des Verfahrens von der Länge des Gradienten $\nabla_w R(w)$ bzw. $\nabla_w R_{\mathcal{I}}(w)$:

(i) In flachen Plateaus mit $\nabla_w R(w) \approx 0$ konvergiert das Verfahren nur langsam

(ii) Ist $\nabla_w R(w)$ sehr groß, können Oszillationen entstehen

Probleme auf den Trainingsdaten: Explodierender Gradient

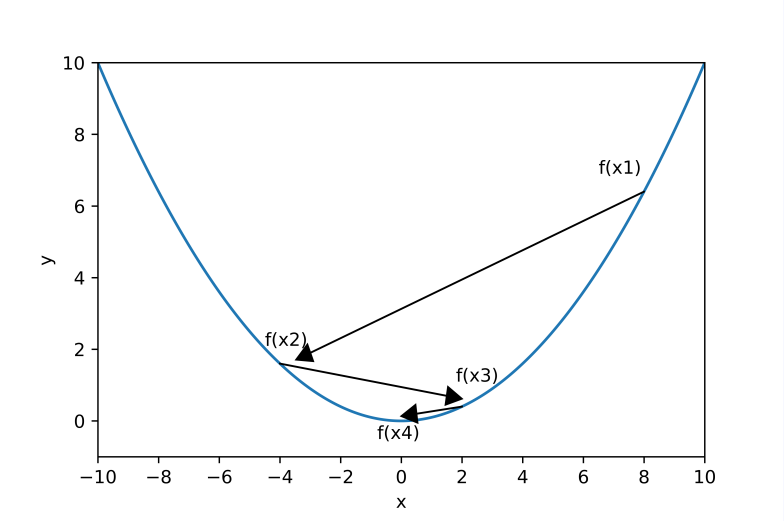
Versuch KNN<64,8> mit SGD-Verfahren auf den Trainingsdaten zu lösen:

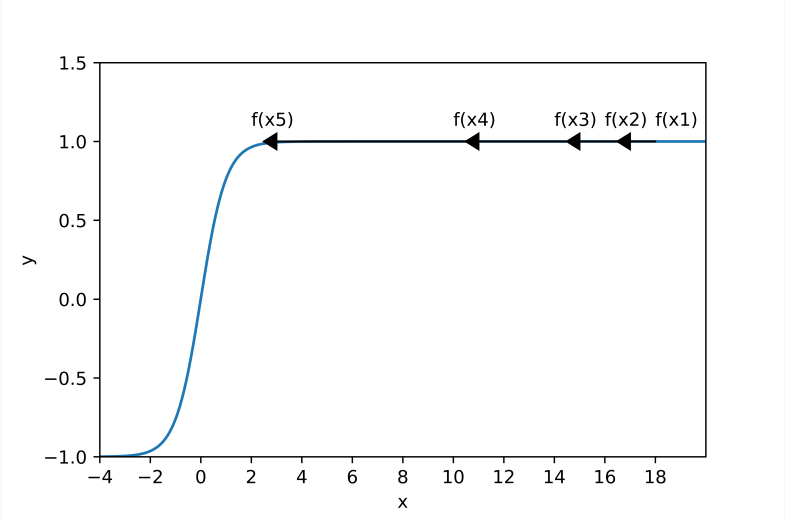
	Kostenfunktion (MSE)	$\|\nabla_w R\|_2$
1. Epoche	1.707.546.112	322.405,90
2. Epoche	1.707.365.888	1.210.910,62
3. Epoche	1.691.739.648	45.805.748,00
4. Epoche	$1{,}35 \times 10^{25}$	∞
5. Epoche	NAN	NAN

Tabelle: Lernrate $\eta = 0{,}000001$

Ziel: Variation des Verfahrens, welches weniger stark von der Lernrate η und der Länge des Gradienten $\nabla_w R$ abhängig ist.

Resilient Backpropagation (Rprop)





Intuition - Entscheide nur durch Vorzeichen des Gradienten in welcher Weise die Parameter verändert werden sollen:

(i) Verringere die Größe der Updates, wenn sich die Abstiegsrichtung verändert

(ii) Erhöhe die Größe der Updates, wenn sich die Abstiegsrichtung nicht verändert

Resilient Backpropagation (Rprop)

Iterationsvorschrift für $w_i \in w$:

$$\xi_i^t = \begin{cases} \min(\xi_i^{(t-1)} \cdot \eta^+, \xi_{\max}), & \text{falls } \nabla_{w_i} R^{(t)} \cdot \nabla_{w_i} R^{(t-1)} > 0, \\ \max(\xi_i^{(t-1)} \cdot \eta^-, \xi_{\min}), & \text{falls } \nabla_{w_i} R^{(t)} \cdot \nabla_{w_i} R^{(t-1)} < 0, \\ \xi_i^{(t-1)}, & \text{sonst} \end{cases}$$

$$w_i^{(t+1)} = w_i^{(t)} + \Delta w_i^{(t)},$$

$$\Delta w_i^{(t)} = -\xi_i{}^{(t)} \text{sgn}(\nabla_{w_i} R^{(t)}) \text{ mit } 0 < \eta^- < 1 < \eta^+.$$

Typische Initialisierung:
$\eta^+ = 1{,}2$; $\eta^- = 0{,}5$; $\xi_{max} = 50$ und $\xi_{min} = 10^{-6}$.

Resilient Backpropagation (Rprop)

Problem von Rprop:

(i) Die Idee hinter dem stochastischen Gradientenverfahren ist, dass durch die Aufsummierung aufeinanderfolgender Mini-Batch Gradienten, die Richtung des echten Gradienten der Fehlerfunktion approximiert wird

(ii) Bei einer Implementation eines Mini-Batch Rprop Algorithmus findet keine Approximation des echten Gradienten mehr statt, da die Updates der Parameter unabhängig von der Länge der Mini-Batch Gradienten sind

Zahlenbeispiel:
Angenommen, ein Gewicht w_i wird durch $\nabla_{w_i} R_{\mathcal{I}}$ aufeinanderfolgend 9 mal um 0,1 und im letzten Schritt um -0,9 geupdatet.
Nach den Updates beträgt die Änderung des Gewichts:

(i) im SGD Verfahren 0

(ii) in einem Mini-Batch Rprop Verfahren
$\eta(\sum_{i=1}^{9} (\eta^+)^i - (\eta^+)^9 \eta^-) \approx 22{,}38\eta$ für $\eta^+ = 1{,}2$ und $\eta^- = 0{,}5$

$\Longrightarrow$ In dieser Form ist die Idee von Rprop nicht als Mini-Batch Verfahren implementierbar.

Adaptive SGD Verfahren

Änderung für stochastische Verfahren:

(i) Normiere Mini-Batch Gradienten mit einem Wert, der für aufeinanderfolgende Gradienten ähnlich groß ist

(ii) Häufig wird hierfür $\sqrt{\mathbb{E}[\nabla_{w_i} R_{\mathcal{I}}^2]}$ verwendet

Update der Parameter für $w_i \in w$ durch:

$$w_i^{(t)} = w_i^{(t)} - \frac{\eta}{\sqrt{\mathbb{E}[\nabla_{w_i} R_{\mathcal{I}}^2]}} \nabla_{w_i} R_{\mathcal{I}}(w^{(t)})$$

für Mini-Batch $\mathcal{I} \subset \{1, \ldots, n\}$.

Adaptive SGD Verfahren

Der Erwartungswert $\mathbb{E}[\nabla_{w_i} R_{\mathcal{I}}^2]$ muss geschätzt werden:

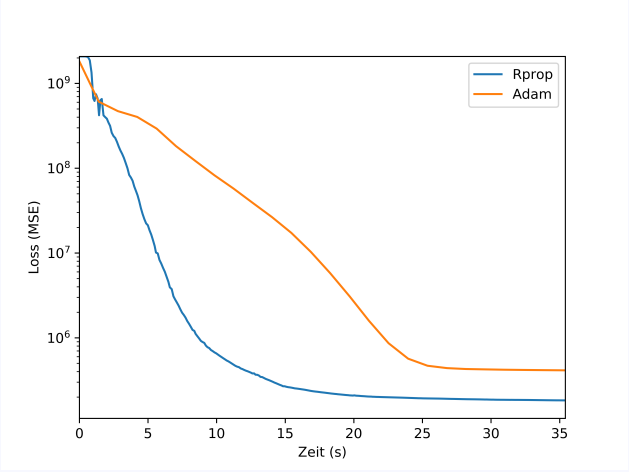
(i) Gewichte bei der Schätzung alle zuvor berechneten Gradienten gleich (Adagrad, siehe [DHS11])

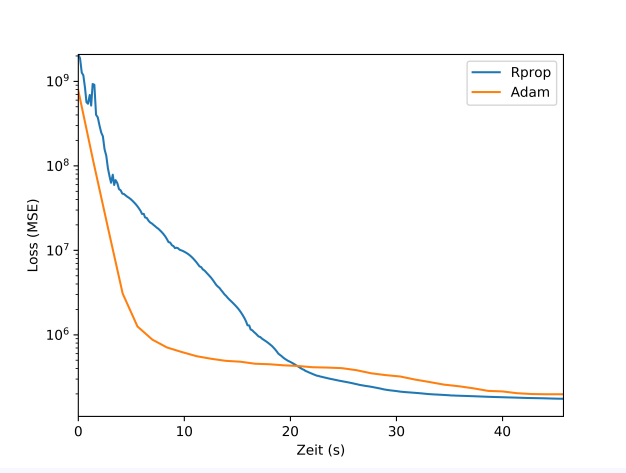
(ii) Gewichte bei der Schätzung die zuletzt berechneten Gradienten schwerer (Adam, siehe [KB14]; RMSprob, siehe [TH12])

Des weiteren wird bei Adam die vorgestellte Technik zusätzlich mit einem Momentum-Term kombiniert. (vgl. Vortrag Einführung in Neuronale Netze)

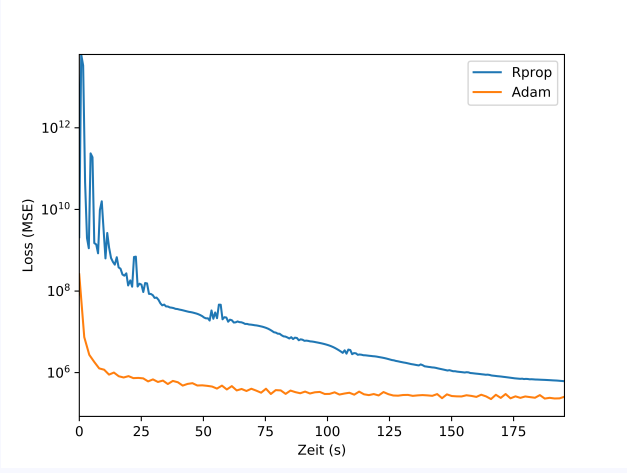
Adam vs Rprop auf den Trainingsdaten

Parameteranzahl $\approx 10^2$





Parameteranzahl $\approx 10^4$



Parameteranzahl $\approx 10^6$

Daten

- Aufteilung in 200.000 Trainings- und 256 Validierungsdatensätze (jeweils Input und Output o1/o3)
- Trainingsdaten eher ungenau
- Company 1: 15 Features pro Datensatz
- Company 2: 16 Features pro Datensatz
- Zielgröße o1: Best Estimate Liabilities
- Zielgröße o3: Eigenmittel/own funds
- wichtig im Rahmen vom Gesetzgeber vorgeschriebenen Regeln, die Versicherungen einhalten müssen (Solvency II)

Daten

Inputs sind Risikofaktoren bzw. deren Veränderung

Component	Risk Factor Description
X_1	Risk-free interest rates movement
X_2	Change in interest rate volatility
X_3	Change in equity volatility
X_4	Shock on volatility adjustment (if used by the company)
X_5	Credit default
X_6	Credit spread widening
X_7	Currency exchange rate risk
X_8	Shock on equity market value
X_9	Shock on property market value
X_{10}	Lapse stress on best estimate assumptions
X_{11}	Mortality catastrophe stress with a one-off increase in mortality
X_{12}	Mortality trend volatility stress
X_{13}	Mortality level stress on best estimate assumptions
X_{14}	Longevity trend volatility stress on best estimate assumptions
X_{15}	Longevity level stress on best estimate assumptions
X_{16}	Morbidity stress on best estimate assumptions
X_{17}	Expenses stress on best estimate assumptions

Verwendete Bibliotheken

- *keras*: Open Source Deep-Learning-Bibliothek mit Schnittstellen zu verschiedenen Frameworks (seit 2015)
- *tensorflow*: Open Source Machine-Learning Framework von Google (seit 2015)

Hyperparameter

- Batch Size
- Anzahl Schichten (hidden layers)
- Anzahl Neuronen pro Schicht
- Aktivierungsfunktion
- Optimierer
- Lernrate
- Anzahl Epochen

Hyperparameter

Batch Size:

- Werte zwischen 64 und 1024
- sollten als Potenz von 2 gewählt werden aufgrund der Rechnerarchitektur

Hidden Layers und Neuronen:

- mehr Hidden Layers können Nicht-Linearität besser darstellen
- Anzahl der Parameter des Netzes sollte Anzahl der Trainingsdaten nicht überschreiten
- Anzahl Hidden Layers: 1-16
- Anzahl Neuronen 2-1024

Aktivierungsfunktionen

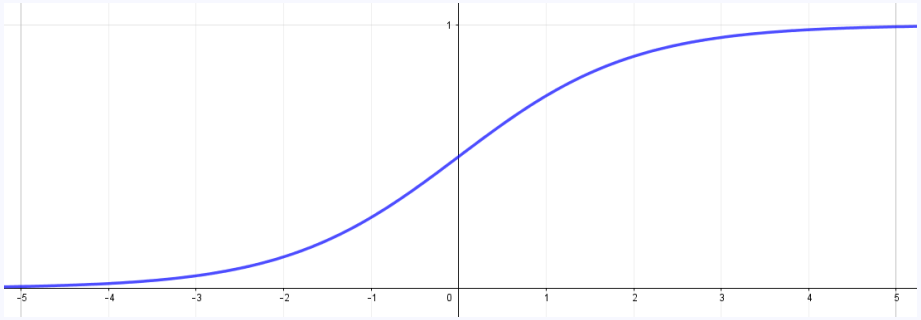


Abbildung: Sigmoid Funktion: $f(x) = \frac{1}{1+\exp(-x)}$

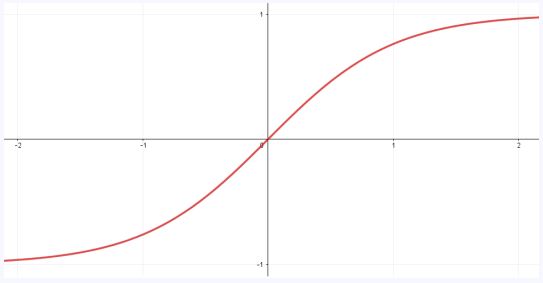


Abbildung: tanh Funktion: $f(x) = \tanh(x)$

Wichtigste Eigenschaft: Nicht-Linear

Warum brauchen wir Alternativen?

Problem des verschwindenden Gradienten

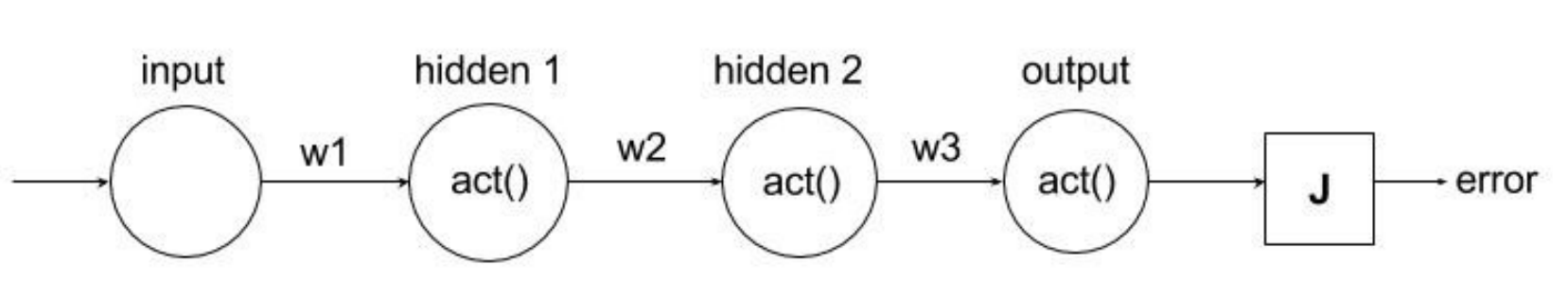


Abbildung: Ein neuronales Netz mit 2 Hidden Layers (Abbilung von https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b)

Warum brauchen wir Alternativen?

Problem des verschwindenden Gradienten

Backpropagation des Fehlers:

$$\frac{\partial Error}{\partial \omega_1} = \frac{\partial Error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial \omega_1}$$

Warum brauchen wir Alternativen?

für die mittleren Terme gilt:

$$z_1 = hidden2 * \omega 3$$

$$\rightarrow \frac{\partial output}{\partial hidden2} = \frac{\partial \tanh(z_1)}{\partial z_1} \omega 3$$

und

$$z_2 = hidden1 * \omega 2$$

$$\rightarrow \frac{\partial hidden2}{\partial hidden1} = \frac{\partial \tanh(z_2)}{\partial z_2} \omega 2$$

Warum brauchen wir Alternativen?

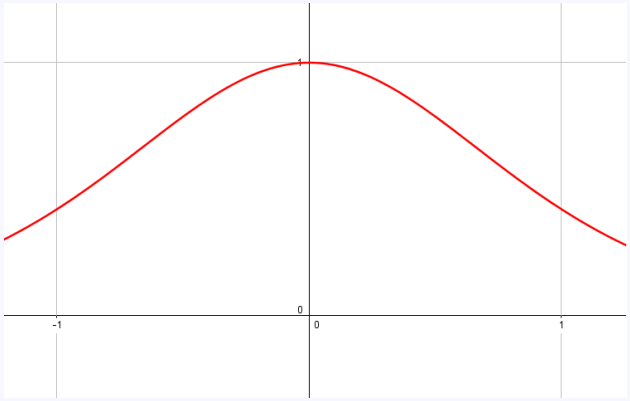


Abbildung: $\tanh(x)' = 1 - \tanh(x)^2 < 1$ für $x \neq 0$

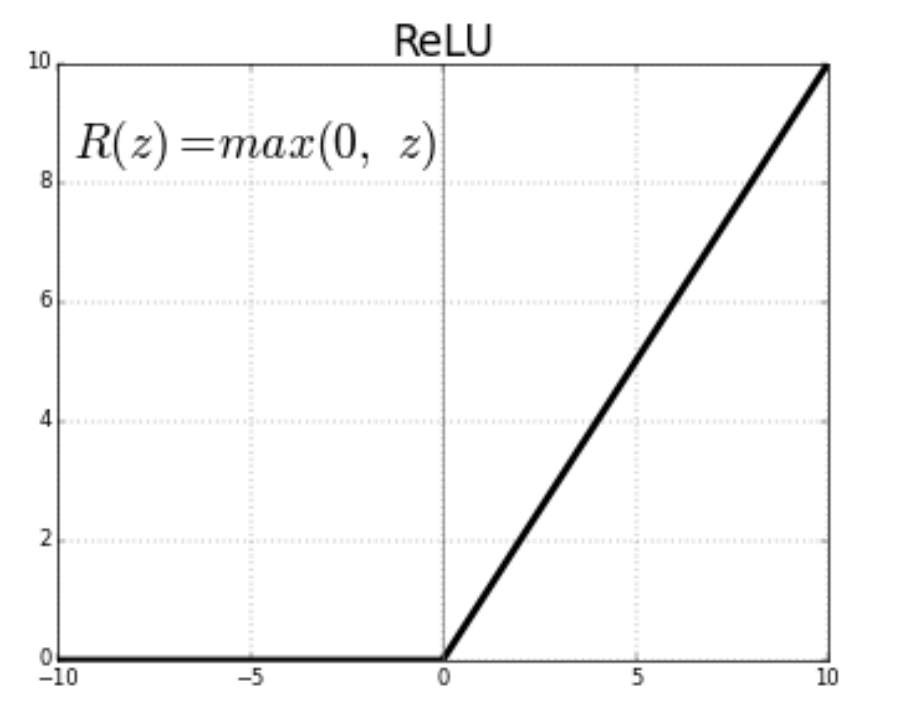
Also:

$$\frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} = \frac{\partial \tanh(z_1)}{\partial z_1}\omega 3 * \frac{\partial \tanh(z_2)}{\partial z_2}\omega 2 << 1$$

$\rightarrow$ Werte zu nahe Null können nicht mehr richtig auf dem Rechner dargestellt werden

$\rightarrow$ der Gradient wird immer kleiner und macht das Training von tiefen Netzen unmöglich

Aktivierungsfunktionen - relu



- ebenfalls nicht-linear, Ableitung entweder 1 oder 0
- einfache Rechenoperation
- kann Model vereinfachen durch 'echte' Nullen
- Training von sehr tiefen Netzwerken möglich

Gewichtsinitialisierung

- Konvergenz des Netzes stark abhängig von Anfangswerten
- Grundsätzlich: zufällige sehr kleine Werte aus Normalverteilung
- je nach Aktivierungsfunktion bestimmte Skalierung um Varianz des Gradienten konstant zu halten

weitere Keras Optionen - EarlyStopping

```
EarlyStopping(monitor='val_loss', min_delta
patience=50)
```

- Abbruch des Trainings bevor alle Epochen durchgelaufen sind
- *monitor* gibt den zu überwachenden Wert an
- *min delta* akzeptable prozentuale Veränderung
- *patience* über wieviele Epochen keine relevante Veränderung auftreten muss, damit abgebrochen wird
- verhindert Overfitting
- steigert Performance, indem sinnlose Optimierungen abgebrochen werden

weitere Keras Optionen - EarlyStopping

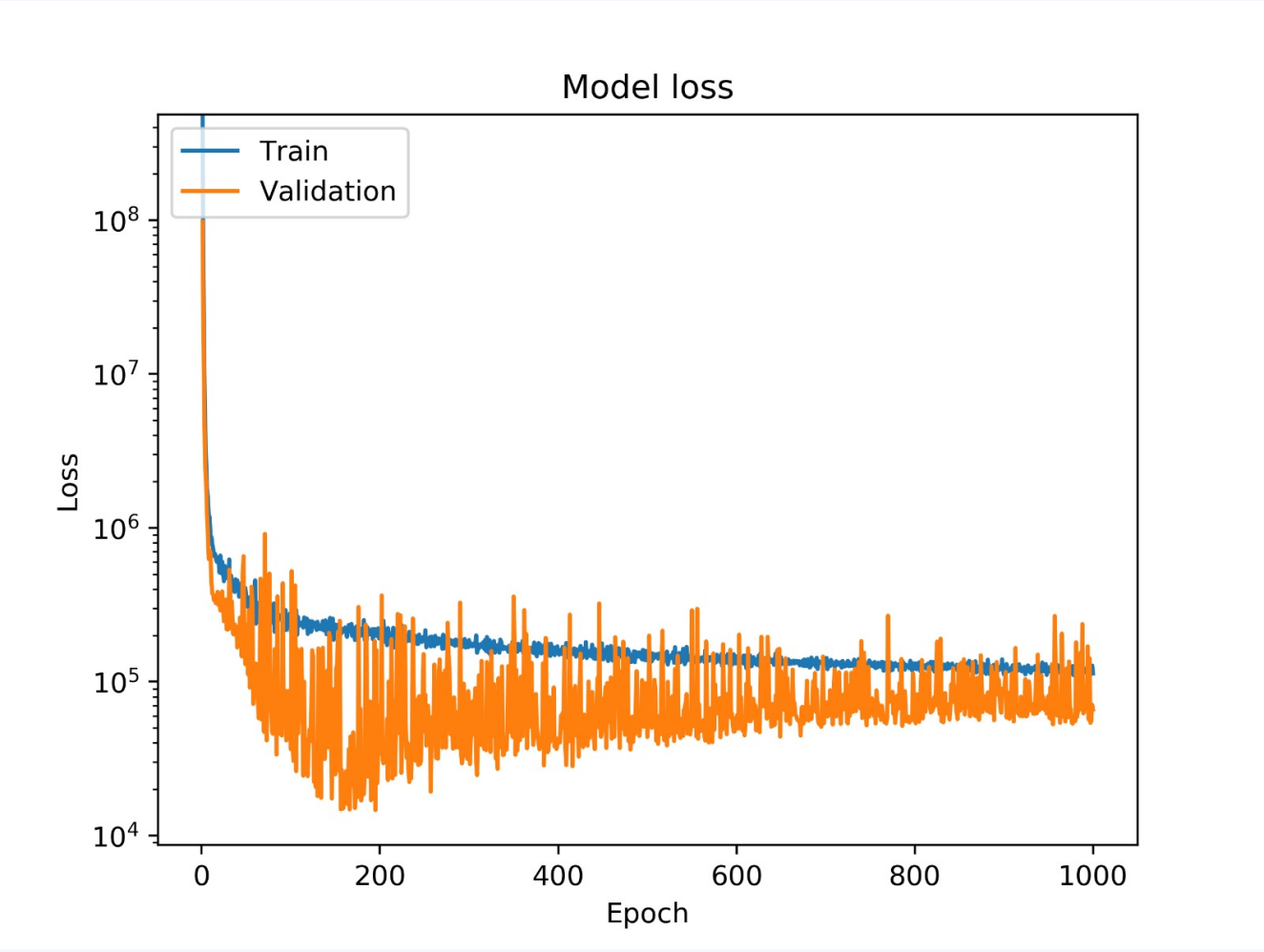


Abbildung: Overfit nach ca. 200 Epochen

weitere Keras Optionen - Checkpoint

```
ModelCheckpoint(checkpoint_name,
monitor='mean_absolute_error',
save_best_only = True)
```

- testet nach jeder Epoche, ob eine Verbesserung in Bezug auf die angegebene loss function aufgetreten ist und speichert die Gewichte in diesem Fall ab
- man erhält immer das beste Netz (auch wenn der Optimierer sich am Ende verrennt)
- Nachteil: kostet Zeit!

Loss Funktion

Mean Absolute Error

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- gewichtet jeden Fehler gleich stark

Mean Squared Error

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- gewichtet große Fehler stärker
- kann bei großen Datensätzen zu groß für die Darstellung auf dem Rechner werden

Loss Funktion

Mean Absolute Percentage Error

$$MAPE = \frac{100\%}{N} \sum_{i=1}^{N} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

- lässt sich leicht interpretieren und vergleichen
- als Loss Funktion jedoch schlecht geeignet

Ergebnisse

Variation der Anzahl an Hidden Layer und Neuronen

Versuchsaufbau:

(i) Wir setzen die Anzahl der Hidden Layer des Netzwerkes konstant auf $k \geq 1$.

(ii) Die Anzahl der Neuronen pro Hidden Layer wird in jeder Iteration verdoppelt: KNN$< 2, k >$, KNN$< 4, k >$,...

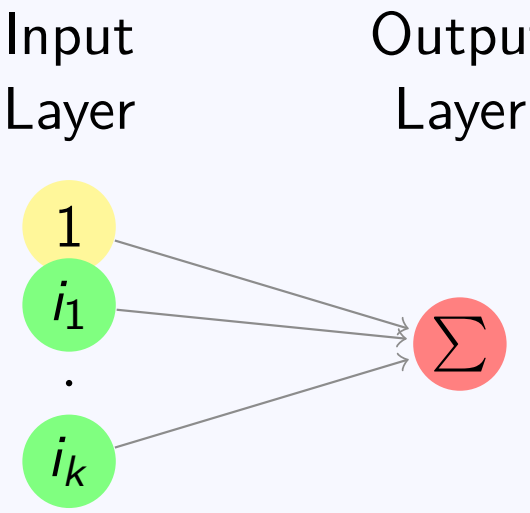
(iii) Aktivierungsfunktion Relu

(iv) Optimierer Rprop und Anzahl der Epochen 4000

Best Estimate Liabilities

Welche Ergebnisse sollten wir mindestens erreichen?

Lineare Regression:



Daten	MSE	MAE	MAPE
Unternehmen 1	255.277, 55	375,73	0,842 %
Unternehmen 2	54.149, 92	162,19	0,398 %

Tabelle: Metriken durch lineare Regression

Unternehmen 1 (Best Estimate Liabilities)

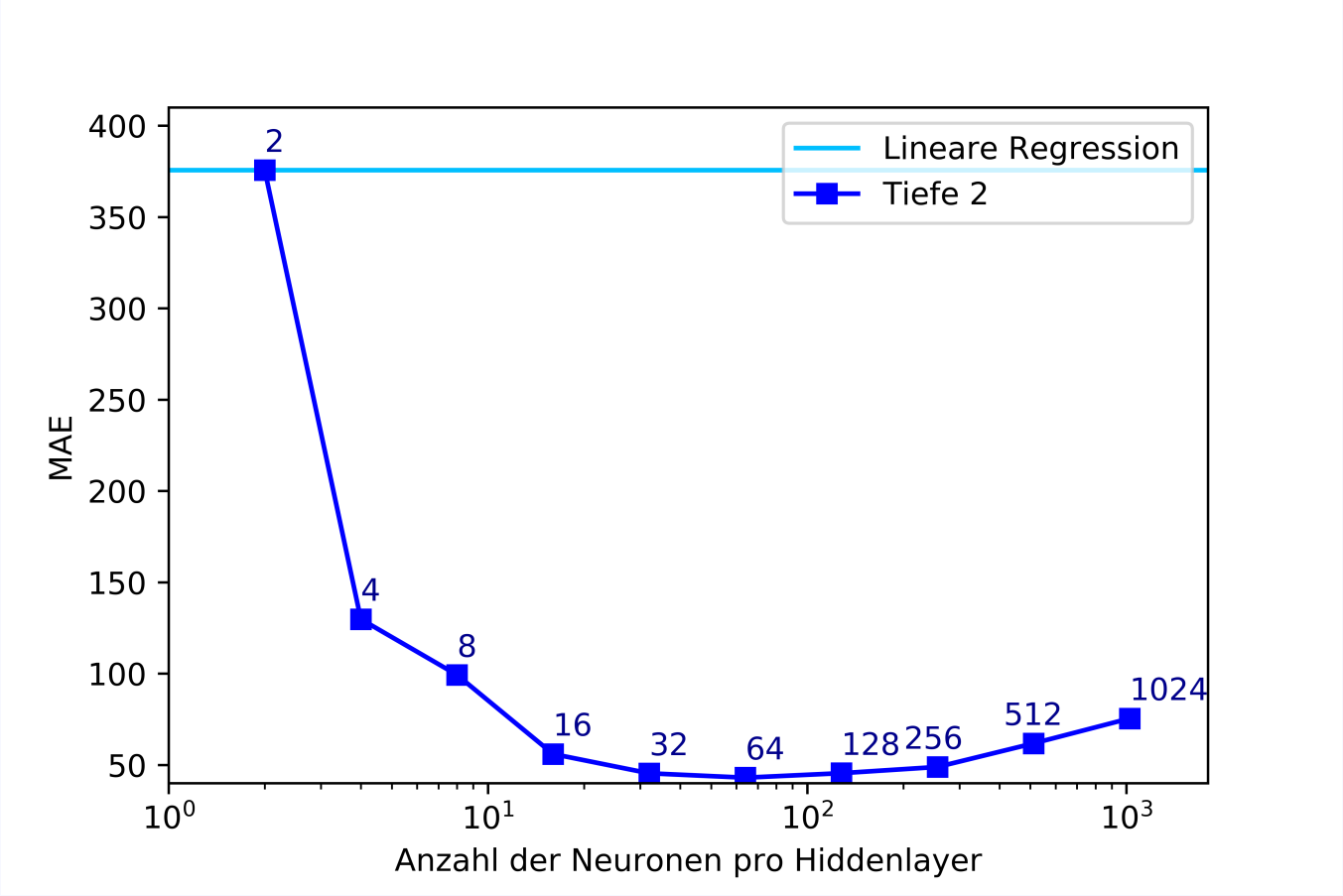


Abbildung: MAE der Validierungsdaten für verschiedene Netzwerke

Unternehmen 1 (Best Estimate Liabilities)

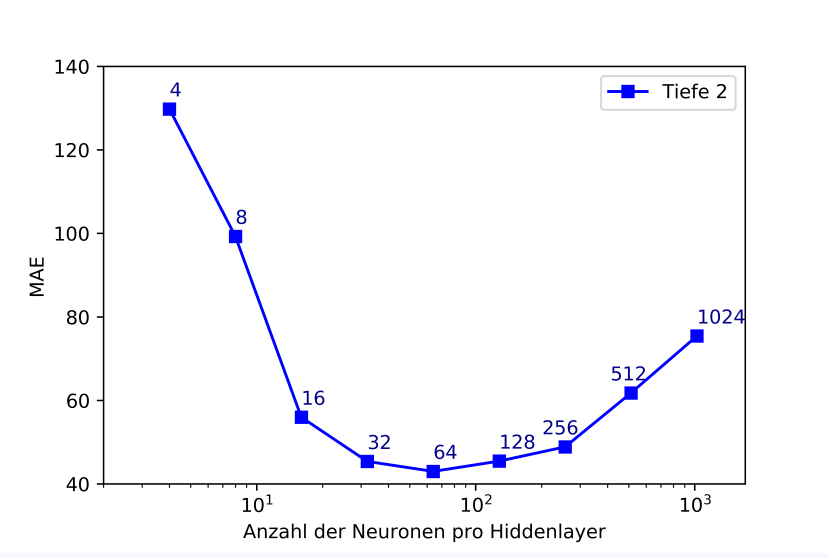


Abbildung: MAE auf den Validierungsdaten

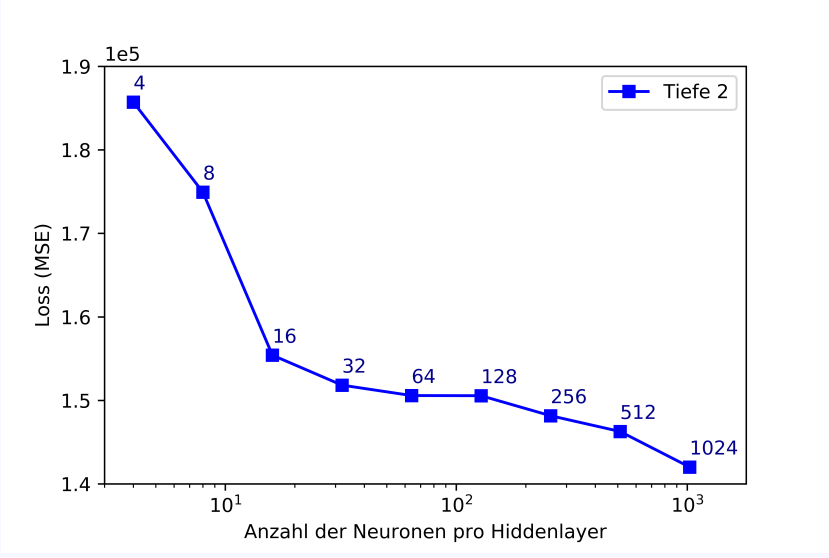


Abbildung: Kostenfunktion auf den Trainingsdaten

Unternehmen 1 (Best Estimate Liabilities)

Erhöhung der Tiefe

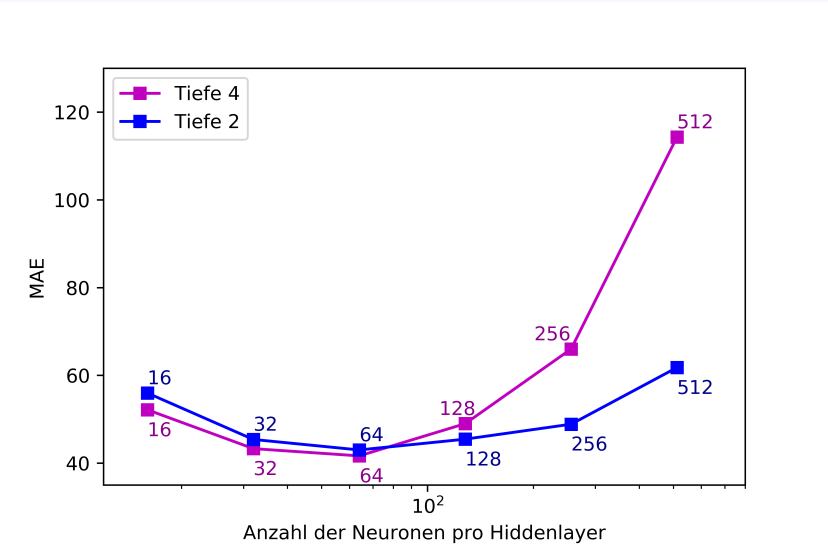


Abbildung: MAE auf den Validierungsdaten

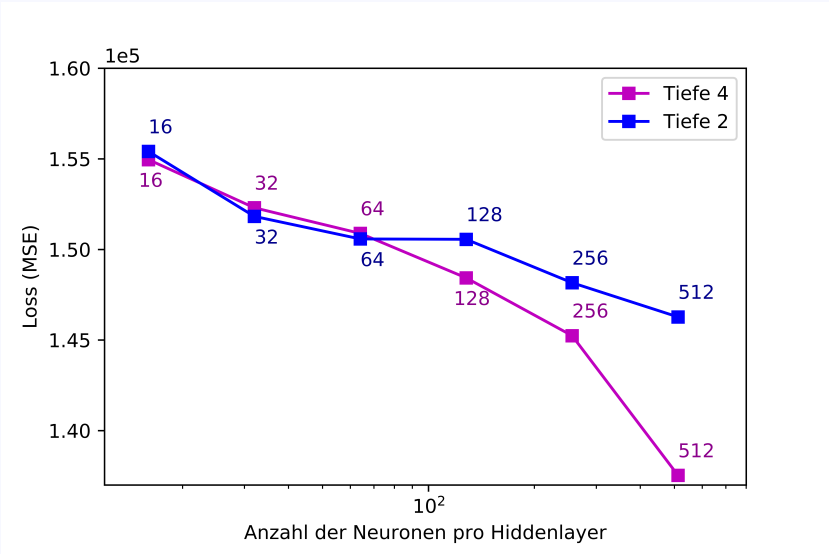


Abbildung: Kostenfunktion auf den Trainingsdaten

Unternehmen 1 (Best Estimate Liabilities)

Erhöhung der Tiefe

Anzahl an Parametern $\approx$ (# Neuronen pro Hidden Layer)^2 $*$ (# Hidden Layer)

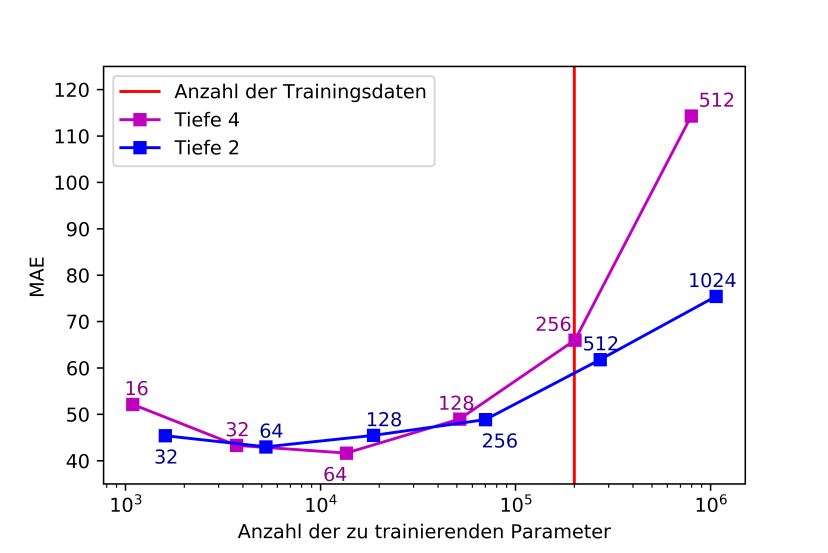


Abbildung: MAE auf den Validierungsdaten

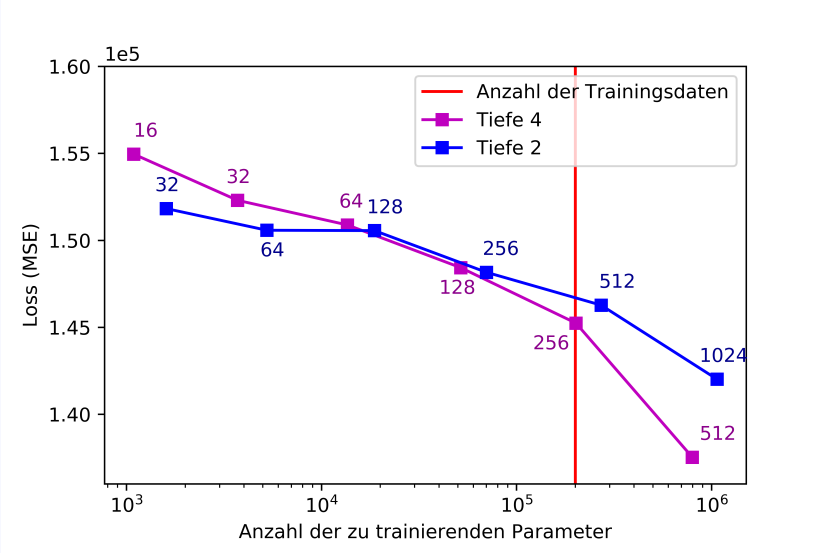


Abbildung: Kostenfunktion auf den Trainingsdaten

Ergebnisse Unternehmen 1 (Best Estimate Liabilities)

Der minimale MAE wird bei Unternehmen 1 bei einem Netzwerk mit 64 Neuronen pro Hidden Layer und 4 Hidden Layer erreicht.

Methode	MSE	MAE	MAPE
Lineare Regression	255.277,55	375,73	0,842 %
Poly. Regression 2	43.620,20	157,53	0,351 %
Poly. Regression 3	8299,72	70,39	0,157 %
Poly. Regression 4	9528,46	75,25	0,164 %
KNN<64,4>	3.200,59	43,29	0,092 %

Tabelle: Ergebnisse Validierungsdaten Unternehmen 1: Vergleich mit anderen Regressionsmodellen

Unternehmen 2 (Best Estimate Liabilities)

Veränderung der Tiefe

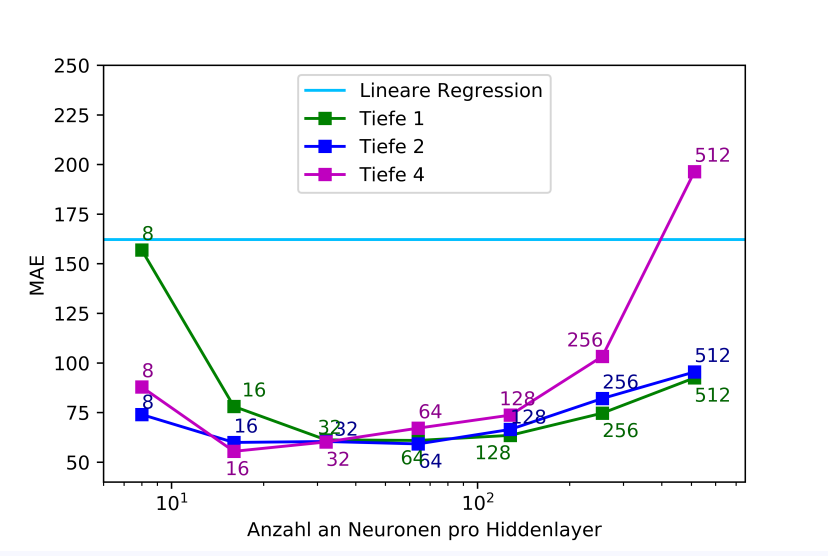


Abbildung: MAE auf den Validierungsdaten

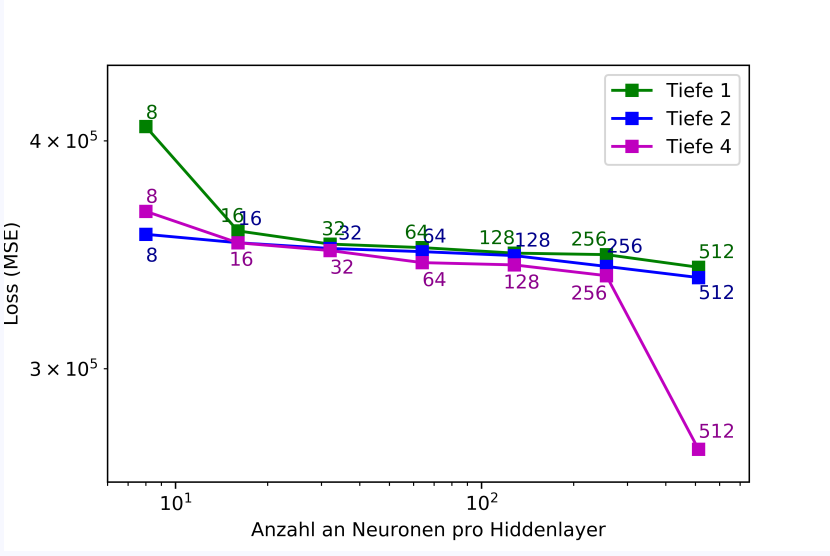


Abbildung: Kostenfunktion auf den Trainingsdaten

Ergebnisse Unternehmen 2 (Best Estimate Liabilities)

Der minimale MAE wird bei Unternehmen 2 bei einem Netzwerk mit 16 Neuronen pro Hidden Layer und 4 Hidden Layer erreicht.

Methode	MSE	MAE	MAPE
Lineare Regression	54.149,92	162,19	0,398 %
Poly. Regression 2	18.925,47	102,69	0,250 %
Poly. Regression 3	8432,45	72,54	0,178 %
Poly. Regression 4	17639,64	101,93	0,247 %
KNN<16,4>	6069,50	55.45	0,136 %

Tabelle: Ergebnisse Validierungsdaten Unternehmen 2: Vergleich mit anderen Regressionsmodellen

Company 1 - Aktivierungsfunktion

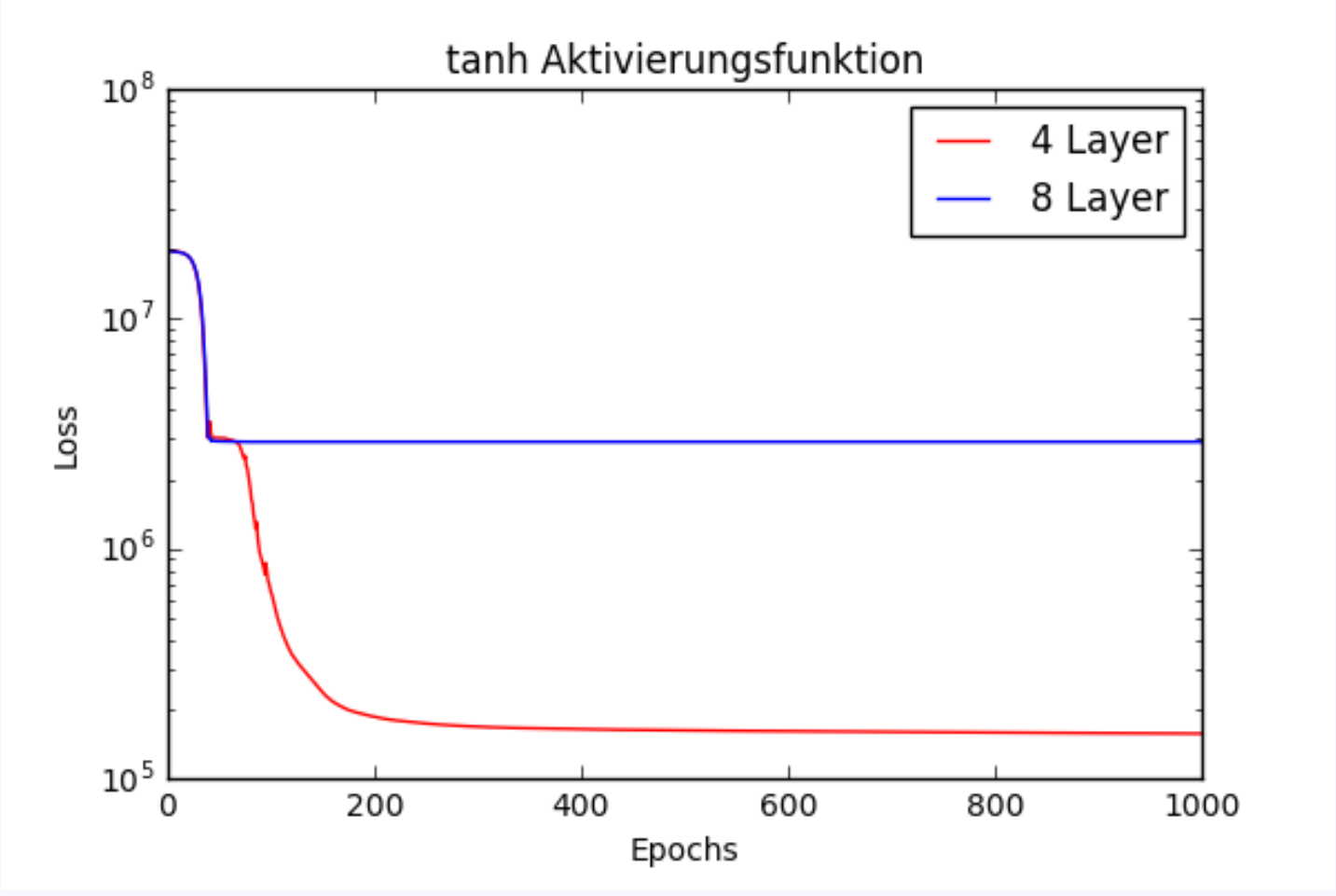


Abbildung: Vergleich tanh für verschieden tiefe Netze

Company 1 - Aktivierungsfunktion

Layer	inital	10 epochs	20 epochs
1	3,3156	0,0003	4,88E-06
2	3,7273	0,0002	4,19E-06
3	3,4440	0,0003	5,06E-06
4	4,7752	0,0005	8,68E-06
5	5,3965	0,0017	2,81E-06
6	5,9384	0,0102	0,0002
7	5,3559	0,3934	0,0110
8	5,7622	130110	509710

Abbildung: Norm des Gradienten

Company 1 - Aktivierungsfunktion

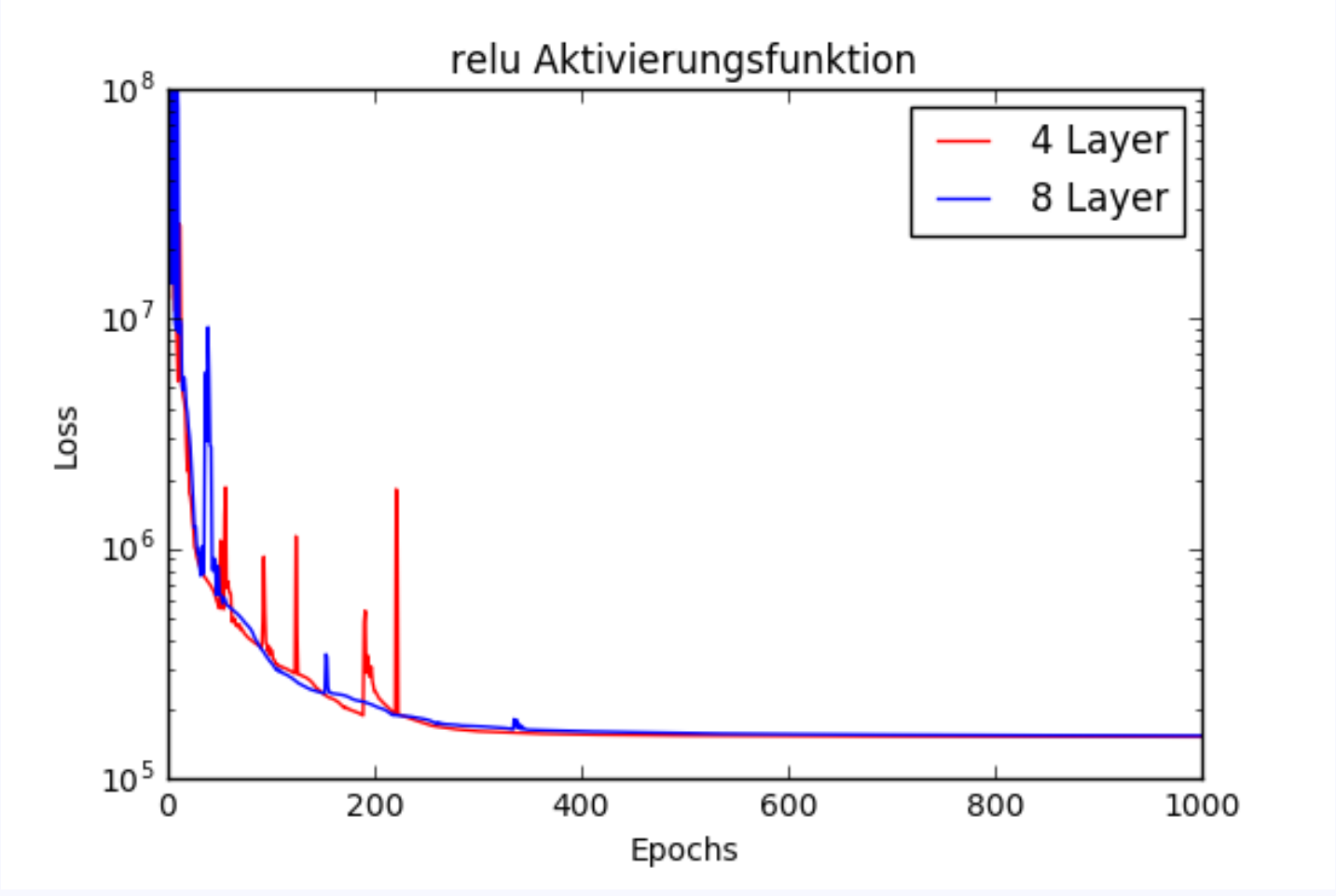


Abbildung: Vergleich relu für verschieden tiefe Netze

Company 1 - Aktivierungsfunktion

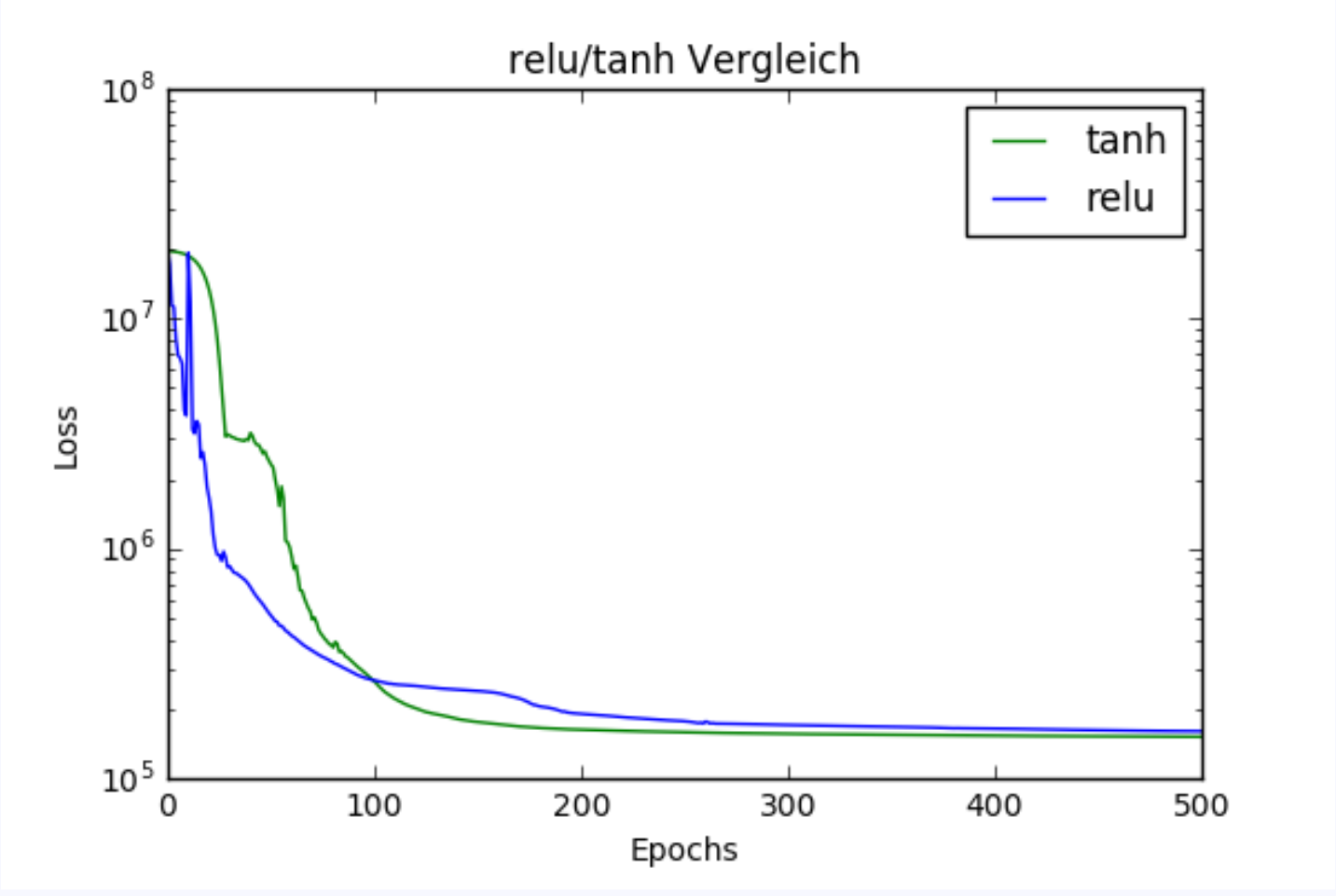


Abbildung: relu und tanh mit 2 Hidden Layers

Company 1 - Aktivierungsfunktion

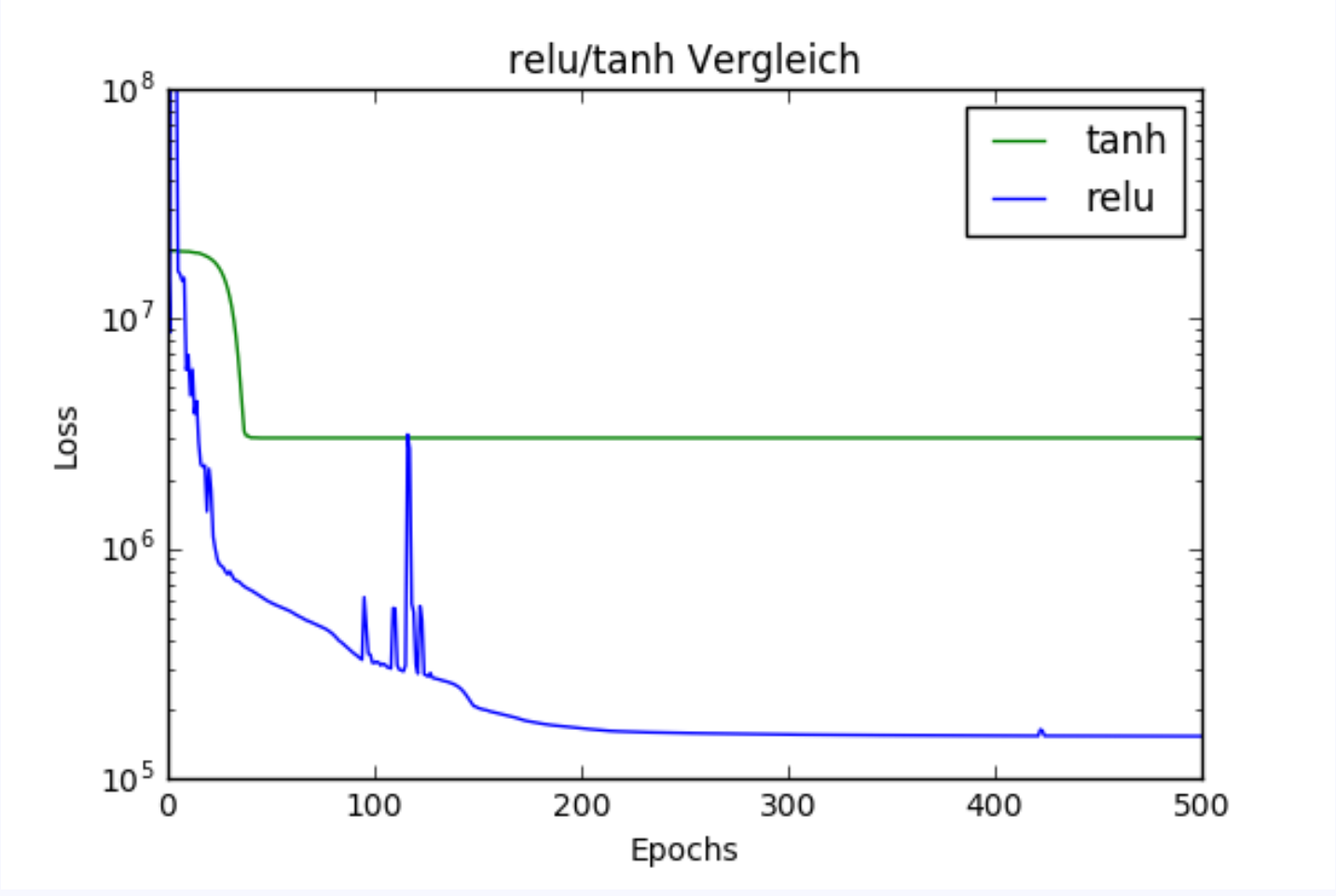


Abbildung: Vergleich relu und tanh mit 8 Hidden Layers

Company 1 Ergebnisse Own Funds

Methode	MSE	MAE	MAPE
Lineare Regression	255428,55	375	30,69 %
Poly. Regression 2	43608,90	157.48	10,62 %
Poly. Regression 3	8315,15	70.47	4,45 %
Poly. Regression 4	9586,20	75,46	4,37 %
KNN<64,4>	3453,16	45,80	2,74 %

Company 1 Ergebnisse Own Funds

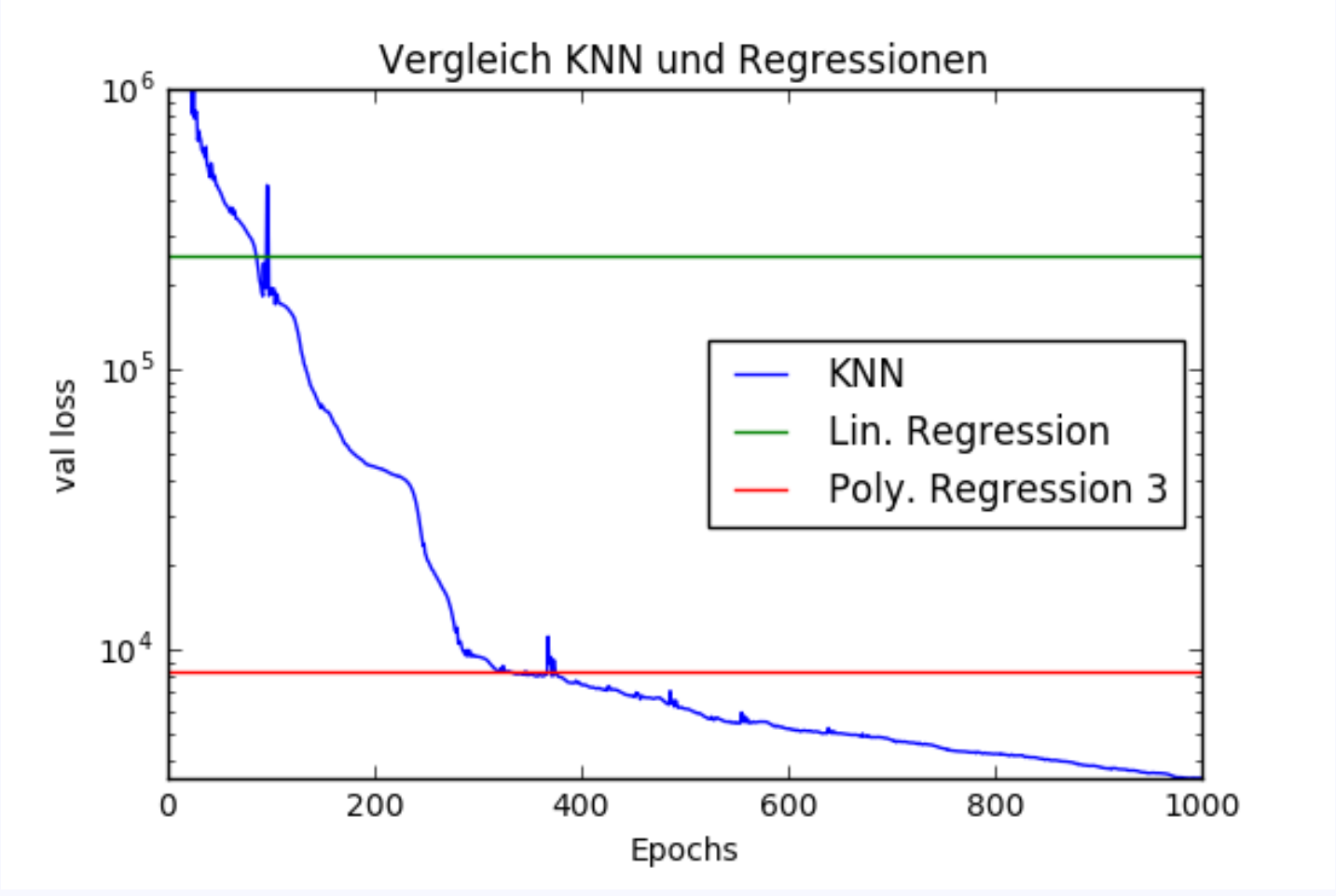


Abbildung: KNN<64,4> schneidet am Besten ab

Company 2 - Batch Size

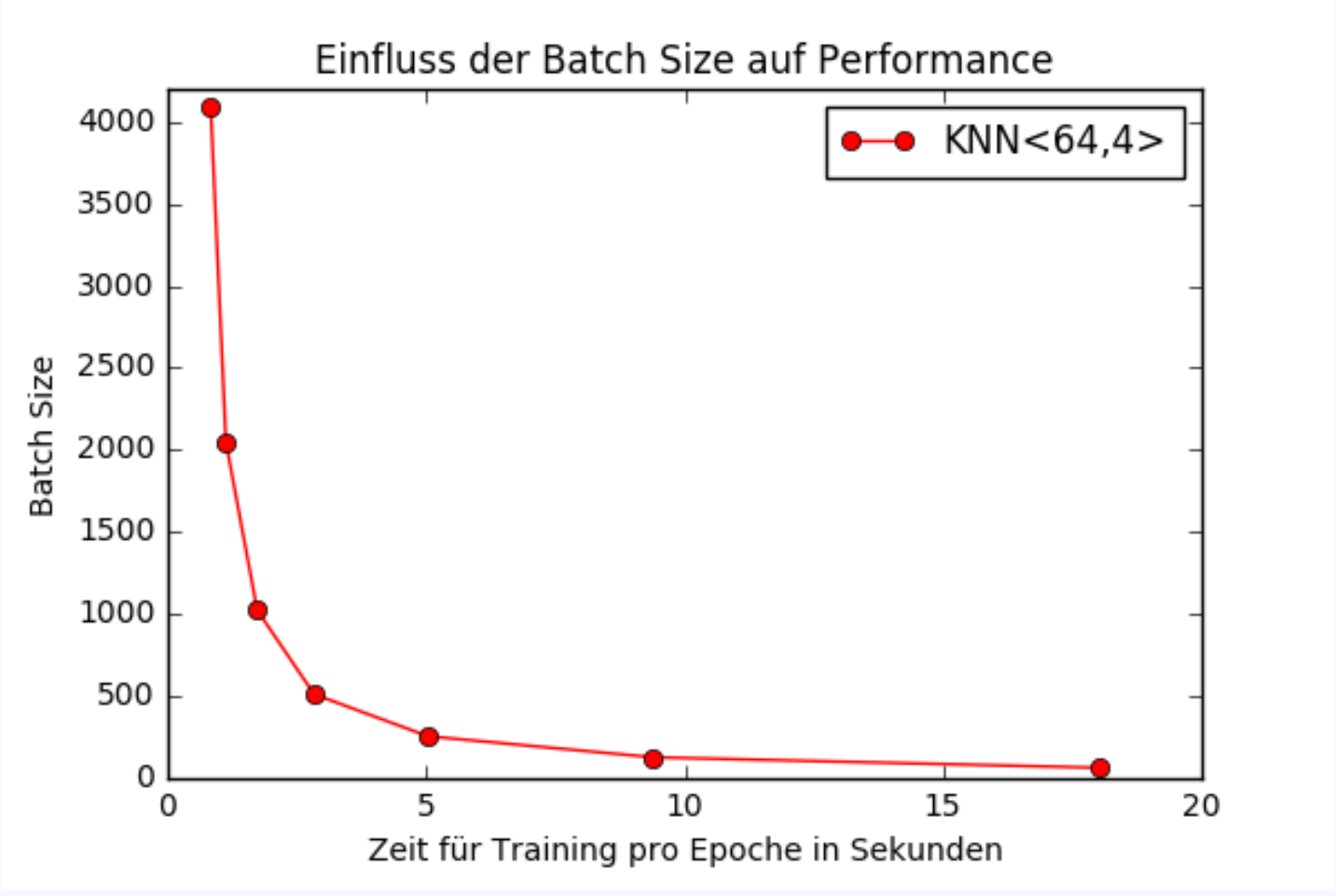
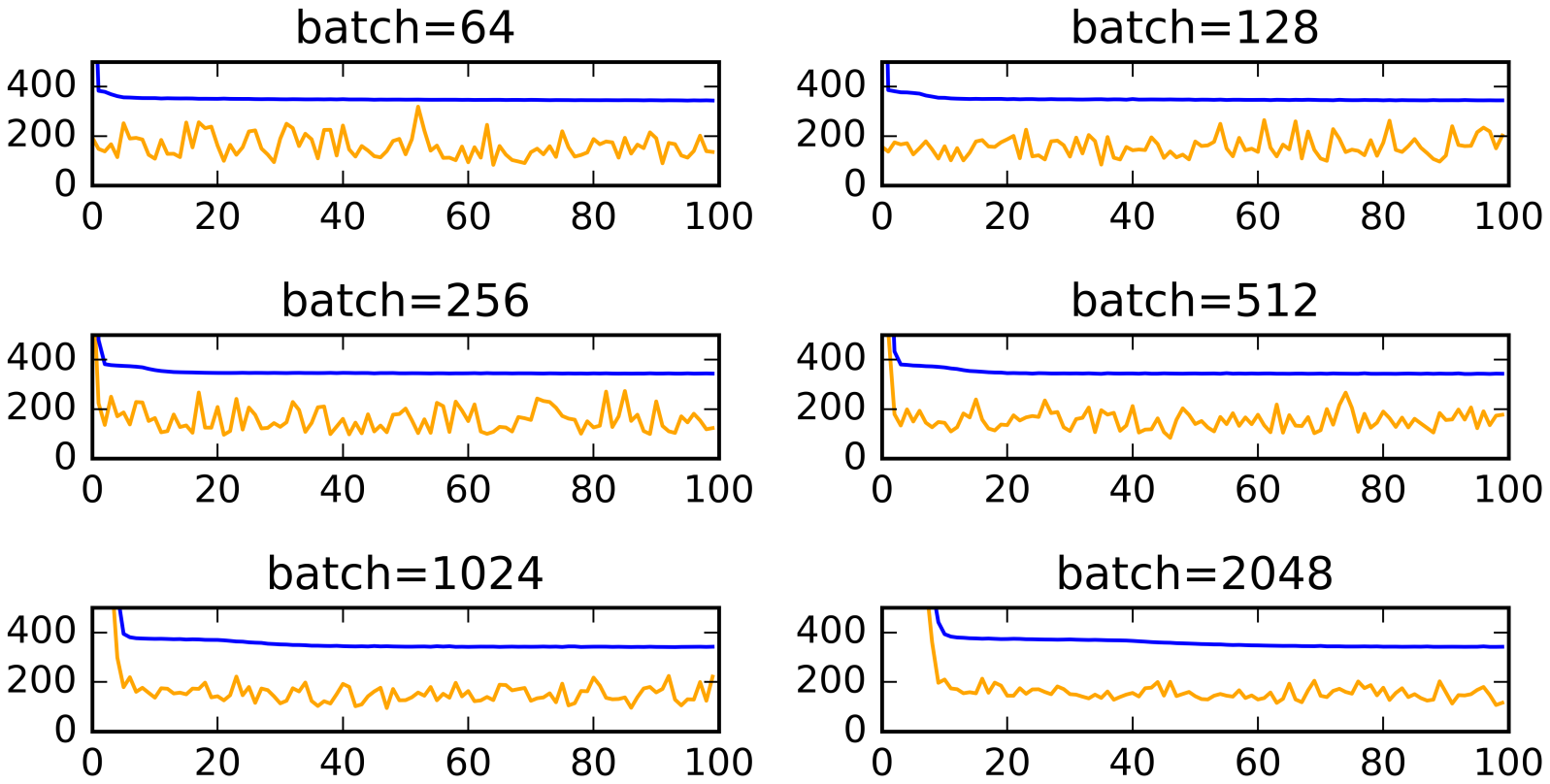


Abbildung: Trade-off zwischen Performance und Batch Size

Company 2 - Batch Size

Abbildung: Konvergenz unterschiedlicher Batch Größen (Trainingsdaten blau, Validierungsdaten orange)



Company 2 Ergebnisse

Methode	MSE	MAE	MAPE
Lineare Regression	54132,91	162,17	2,49 %
Poly. Regression 2	18929,06	102,70	1,55 %
Poly. Regression 3	8433,70	72,55	1,08 %
Poly. Regression 4	17642,36	101,94	1.46 %
KNN<64,4>	6505,84	60,05	0,89 %

Company 2 Ergebnisse Own Funds

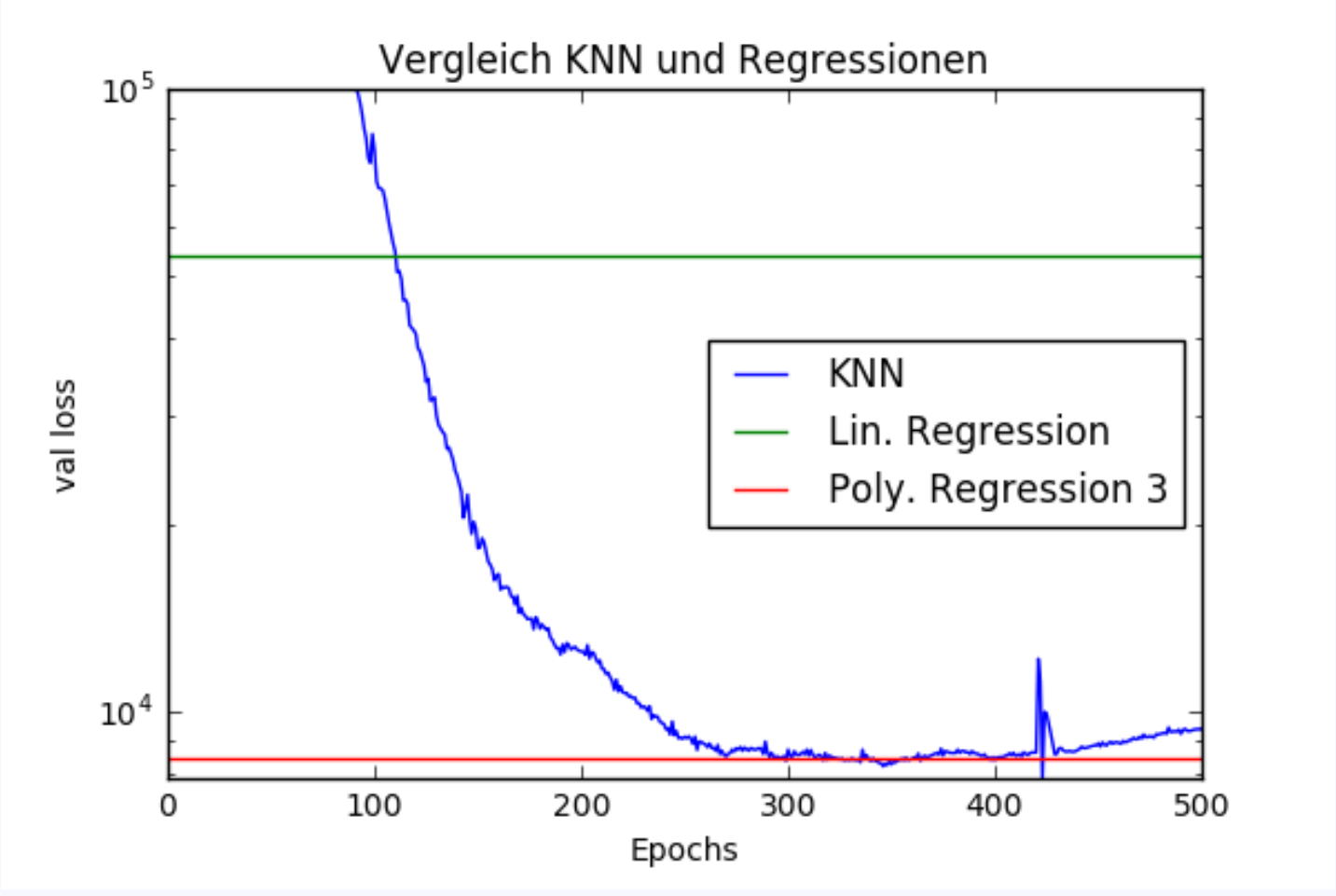


Abbildung: KNN<64,4> schneidet am Besten ab

Zwischenfazit

- Neuronale Netze liefern in allen Fällen bessere Ergebnisse als andere Regressionen
- Verbesserungen von bis zu XY 88,47 % im Vergleich zu einer Linearen Regression
- für Company 1 größere Verbesserung als bei Company 2
- **aber:** höherer Programmier- und Rechenaufwand

Ensemble

Ensemble

Bilde zufällig $m \in \mathbb{N}$ Netzwerke, die die Mindestanforderung erfüllen:

(i) Anzahl der zu trainierenden Parameter darf die Anzahl der Trainingsdaten nicht überschreiten

(ii) Fehler darf nicht schlechter als der einer linearen Regression sein

Kombiniere ihre Schätzungen:

(i) Sei $\hat{y}_i$ der Vektor, den wir als Schätzung durch die Daten x von Netzwerk $i \in 1, \ldots, m$ erhalten

(ii) Betrachte Teilmenge $\mathcal{A} \subseteq \{1, \ldots, m\}$ an Netzwerken und setze $\hat{y}_{new} = \frac{1}{|\mathcal{A}|} \sum_{j \in \mathcal{A}} \hat{y}_j$

10 zufällige Netzwerke (Beispiel)

#	KNN	Optimierer	Batch-Size	Aktivierungsfunktion	MAE
1.	KNN<128,2>	Adam	512	Sigmoid	42,86
2.	KNN<64,1>	Rprop	-	Relu	51,01
3.	KNN<64,1>	Adam	256	Tanh	51,85
4.	KNN<8,2>	Adam	256	Sigmoid	59,94
5.	KNN<32,8>	Adam	256	Relu	62,84
6.	KNN<128,8>	Adam	128	Relu	66,89
7.	KNN<128,1>	Adam	1024	Relu	80,73
8.	KNN<8,4>	Rprop	-	Relu	118,12
9.	KNN<8,2>	Rprop	-	Relu	121,93
10.	KNN<16,1>	Adam	128	Relu	364,21

Tabelle: 10 zufällig ausgewählte Netzwerke, MAE auf den Validierungsdaten Unternehmen 1 Best Estimate Liabilities

Kombination von 3 Netzwerken (Beispiel)

#	KNN	Optimierer	Batch-Size	Aktivierungsfunktion	MAE
1.	KNN<128,2>	Adam	512	Sigmoid	42,86
2.	KNN<64,1>	Rprop	-	Relu	51,01
3.	KNN<64,1>	Adam	256	Tanh	51,85

Tabelle: 3 Netzwerke der 10 zufällig ausgewählten Netzwerke

Kombination	Best MAE_{single}	MAE_{neu}	Fehler Reduktion
(1,2)	42,86	39,82	7,09 %
(1,3)	42,86	39,56	7,69 %
(2,3)	51,01	44,06	13,62 %
(1,2,3)	42,86	38,48	10,21 %

Tabelle: Neuer Fehler der Kombination auf den Validierungsdaten, Reduzierung des Fehlers im Vergleich zum besten in dieser Kombination erhaltenen Netzwerk

Wieviele Netzwerke sollten wir kombinieren?

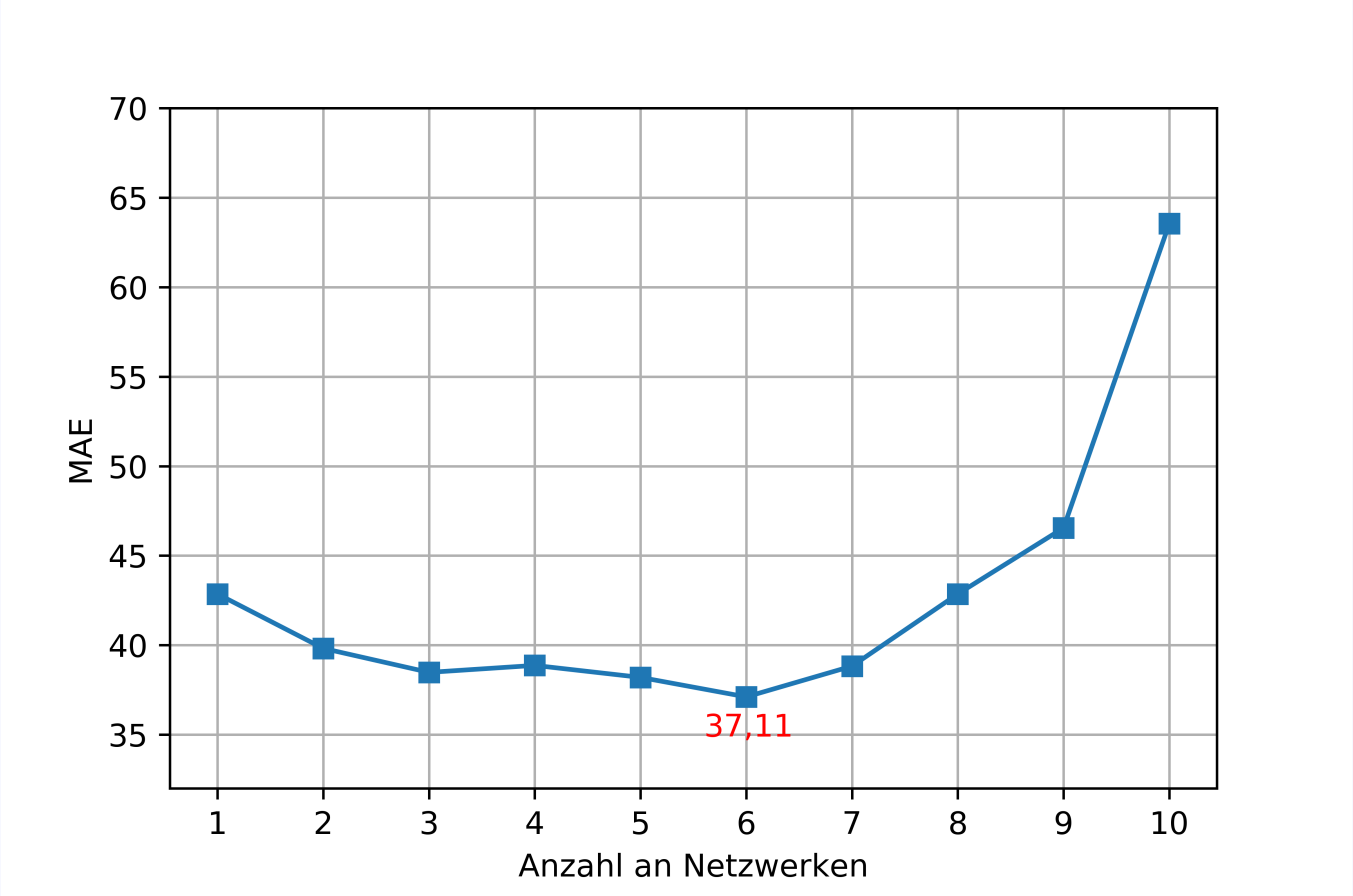


Abbildung: Veränderung des Fehlers; Start mit dem besten Netzwerk und aufeinanderfolgendes Hinzufügen des nächstbesten, nicht in der Kombination erhaltenem Netzwerk

Verbesserung der Ergebnisse durch Ensemble

Daten	MAE Solo	MAE Ens.	Fehler Red.
Unternehmen 1, o1	42,86	37,11	10,21 %
Unternehmen 2, o1	55,45	45,45	18,03 %
Unternehmen 1, o3	45,80	36,60	20,08 %
Unternehmen 2, o3	60,05	47,23	21,34 %

Tabelle: Reduktion des Validierungsfehlers für alle Datensätze

Warum kann sich durch die Kombination von Netzwerken das Ergebniss verbessern?

(i) Durch das Variieren von Hyperparametern, durch die zufällige Initialisierung der Gewichte und auch durch die Verwendung stochastischer Verfahren, werden verschiedene Minima erreicht und so können verschiedene Aspekte des Problems dargestellt werden

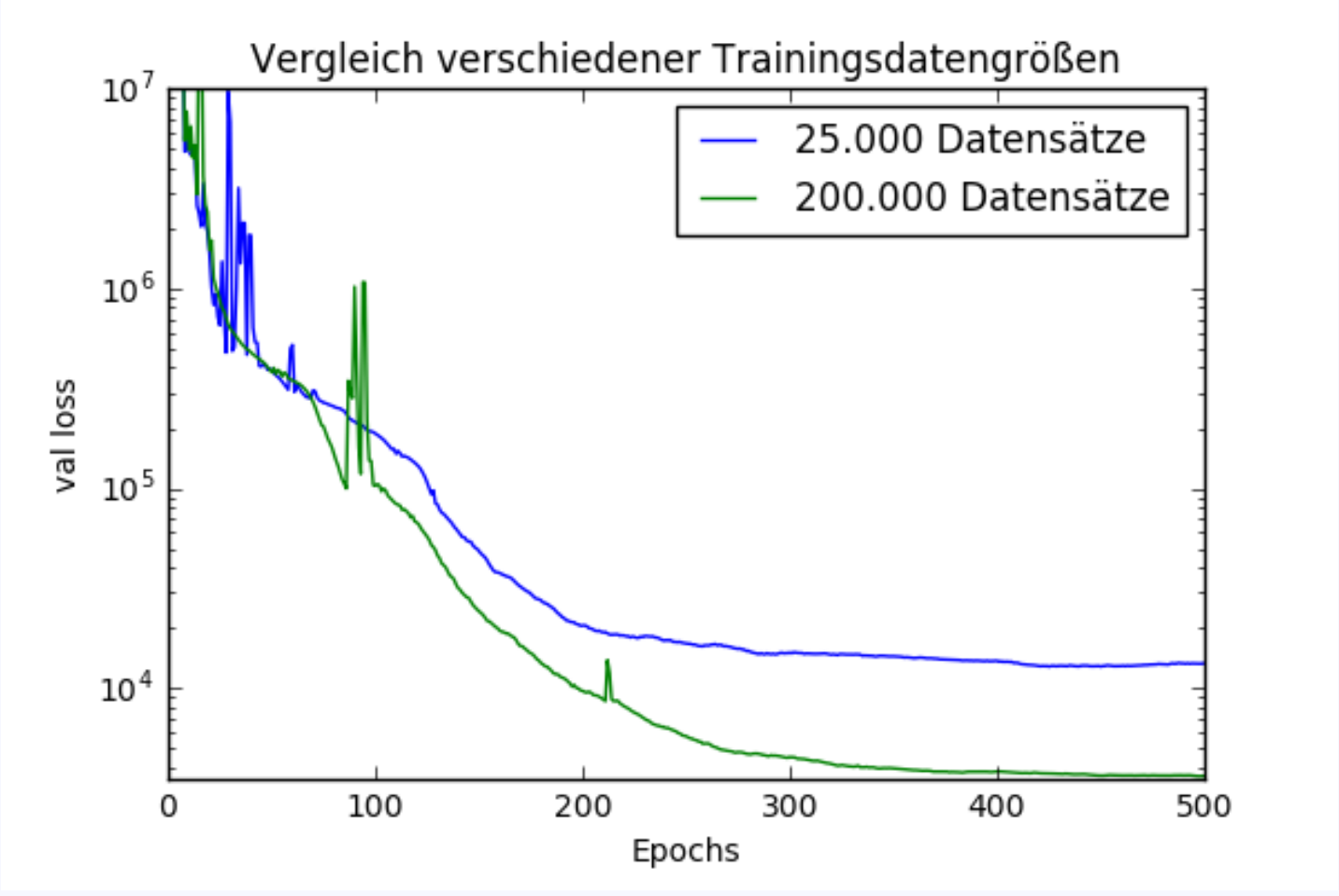
(ii) Wenn nun die verschiedenen Modelle nicht den gleichen Fehler machen, kann so eine bessere Generalisierung erreicht werden

Performance

Performance steigern

- Pruning
- weniger Daten nutzen (z.B. nur 25.000)
- größere Batch Size
- auf Grafikkarte (GPU) rechnen

weniger Trainingsdaten nutzen



Warum Grafikkarten?

- eigentlich für grafische Darstellung verwendet
- ausgelegt viele Rechnungen parallel auszuführen
- Deep Learning braucht viele Matrix-Vektor Multiplikationen

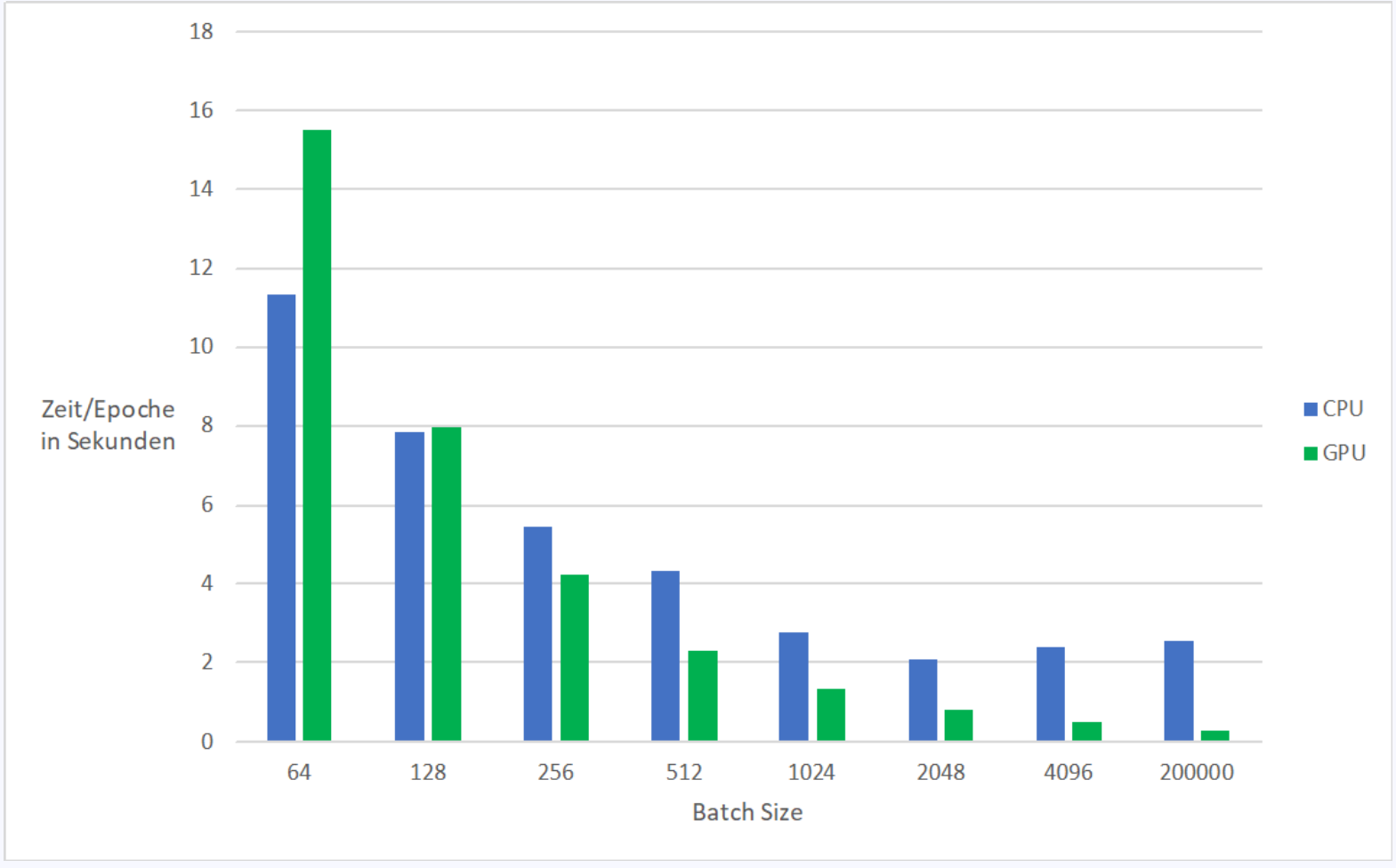
CPU vs GPU

Batch Size	CPU Zeit/Epoche	GPU Zeit/Epoche
64	11,34 s	15,52 s
128	7,85 s	7,97 s
256	5,44 s	4,22 s
512	4,31 s	2,28 s
1024	2,77 s	1,32 s
2048	2,07 s	0,79 s
4096	2,38 s	0,51 s
200000	2,55 s	0,25 s

Abbildung: Vergleich Laufzeiten CPU und GPU

CPU vs GPU

Abbildung: Vergleich Laufzeiten CPU(Intel Dual Core @2,70 GHz) und GPU (Nvidia Tesla K80)



Literatur

Literatur I

Imad Dabbura.
Gradient Descent Algorithm and Its Variants.
https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3
2017.
[Eingesehen: 25.06.19].

John Duchi, Elad Hazan, and Yoram Singer.
Adaptive subgradient methods for online learning and stochastic optimization.
J. Mach. Learn. Res., 12:2121–2159, July 2011.

Literatur II


Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.


Christian Igel and Michael Hüsken.
Improving the rprop learning algorithm.
https://pdfs.semanticscholar.org/df9c/6a3843d54a28138a596acc85a96367a064c2.pdf, 2000.


Diederik Kingma and Jimmy Ba.
Adam: A method for stochastic optimization.
International Conference on Learning Representations, 12 2014.

Literatur III


T. Tieleman and G. Hinton.
Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning.
https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2012.
[Eingesehen: 25.06.19].