Inhalt

1 Einleitung

Funktionalgleichungen benötigen individuelle algebraische/analytische Lösungsverfahren, sodass ein geschlossener Algorithmus nicht entwickelbar ist und sich eine numerische Lösung mittels Machine Learning anbietet, da jede stetige Funktion auf einer kompakten Teilmenge des $\mathbb{R}^n$ sich durch ein tiefes Netz mit ReLU Aktivierung approximieren lässt. (Hornik, Stinchcombe and White, 1989)

Für diese Hausarbeit wurden vier Funktionalgleichungen mit partikulärer Lösung und eine nicht lösbare Funktionalgleichung definiert, damit garantiert ist, dass das MLP keine trivialen Lösungen oder Mischformen von Teillösungen einer allgemeineren Funktionalgleichung lernt. Es wird jeweils ein MLP, also ein vollverbundenes, mehrschichtiges Netz aus linearen Neuronen $g(W^T\text{x+b})$ (Goodfellow, Bengio and Courville, 2016) mit ReLU oder Tanh Aktivierung, implizit darauf trainiert, die gewünschte Funktion zu erlernen.

2 Analytische Lösung der Funktionalgleichungen

2.1 Funktionalgleichung I (Cauchy additiv)

$$Finde\ alle\ stetigen\ Funktionen\ f: \mathbb{R}+ \rightarrow \mathbb{R}+\ mit\ 1.\ f(x+y) = f(x) + f(y)\ und\ 2.\ f\big(f(x)\big) = 5x$$

$I. Wir\ wollen\ zeigen, dass\ f\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} f(x_i)\ f\ddot{u}r\ alle$ n $\in \mathbb{N}+$

$F\ddot{u}r$ n $= 2$ ist die Aussage offensichtlich wahr. Wir setzen dies ein beliebiges für n $\in \mathbb{N}$ voraus.

$Induktionsschritt$: $f(\sum_{i=1}^{n+1} x_i) = f\big((\sum_{i=1}^{n} x_i) + x_{n+1}\big) = f(\sum_{i=1}^{n} x_i) + f(x_{n+1}) = (\sum_{i=1}^{n} f(x_i)) + x_{n+1}$

$II. Hieraus\ folgt\ f\ddot{u}r\ eine\ Konstante$ c $\in \mathbb{N}+$: $f(c * x) = c * f(x)$

$III. Sei\ x = \frac{a}{b}\ t, wobei\ \frac{a}{b} \in \mathbb{Q}+$. Dann gilt: f(x ∗ b) = f(a ∗ t) = b ∗ f(x) = a ∗ f(t) = b ∗ $f\left(\frac{a}{b} * t\right)$ $Und\ somit$ $f\left(\frac{a}{b} * t\right) = \frac{a}{b} * f(t)$

Sei f(1) = c . $Dann\ gilt\ f(x * 1) = x * c = f(x)$

$IV. Wir\ zeigen, dass f(x) = x * c\ stetig\ ist\ und\ somit\ III.\ auf\ \mathbb{R}+\ erweiterbar\ ist.$

$$Sei\ |x - x_0| < \delta\ und\ \delta = \frac{\varepsilon}{c}: \ |f(x) - f(x_0)| = |cx - cx_0| = c|x - x_0| \leq c * \delta = \varepsilon$$

$$Sei\ x \in \mathbb{R}+\ \text{und}\ \sum_{i}^{\infty} x_i\ die\ Dezimaldarstellung\ von\ x. Da\ f\ stetig\ ist\ gilt:$$

$$\sum_{i=1}^{\infty} f(x_i) = f(\sum_{i=1}^{\infty} x_i)$$

Aus IV eingesetzt in Bedingung Nr. 2 folgt: $f\big(f(x)\big) = f(c * x) = c^2 * x = 5x$.

Also $f(x) = \sqrt{5} * x, was\ durch\ Einsetzen\ in\ die\ FG\ leicht\ verifiziert\ werden\ kann.$

2.2 Funktionalgleichung II (Cauchy exponentiell)

$$Finde\ alle\ stetigen\ Funktionen\ f: \mathbb{Q} \to \mathbb{Q}\ mit\ 1.\ f(x+y) = f(x) * f(y)\ und$$

$$2.\ f(2) = 67$$

$I.\ y \to 0: f(2) = f(2) * f(0), also\ f(0) = 1$

$II.\ f(x) = f\left(\frac{x}{2} + \frac{x}{2}\right) = f\left(\frac{x}{2}\right)^2\ also\ f(c) > 0\ für\ alle\ c \in \mathbb{Q}$

$III.\ Bedingt\ durch\ II.\ dürfen\ wir\ logarithmieren.\ Sei\ g(x) = \ln\big(f(x)\big)$:

$$g\big(f(x+y)\big) = g(x+y) = \ln\big(f(x)\big) = \ln\big(f(x) * f(y)\big) = \ln\big(f(x)\big) + \ln\big(f(y)\big)$$
$$= g(x) + g(y).$$

$Wir\ wissen\ durch\ die\ erste\ Funktionalgleichung, dass\ g(x) = c * x.$

$\ln\big(f(x)\big) = c * x\ \ also\ f(x) = e^{cx}$

$Aus\ Bedingung\ Nr.\ 2\ folgt\ f(x) = 67^{\frac{x}{2}}.$

$Stetigkeit: Sei\ x_i\ eine\ beliebige\ Folge\ natürlicher\ Zahlen\ mit\ \lim_{i\to\infty} x_i =$

$x.\ Dann\ gilt\ \lim_{i\to\infty} f(x_i) = \lim_{i\to\infty} \sqrt{67}^{x_i} = \sqrt{67}^{x} = \sqrt{67} * \ldots * \sqrt{67} = f(1)^x = f(x) =$

$f(\lim_{i\to\infty} x_i).$

$Sei\ \ y_i\ eine\ beliebige\ Nullfolge\ und\ x_i\ eine\ beliebige\ Folge\ natürlicher\ Zahlen\ mit\ \lim_{i\to\infty} x_i =$

$x.\ Dann\ gilt\ \lim_{i\to\infty} f(y_i) = \lim f(x_i - x) = \lim_{i\to\infty} \sqrt{67}^{x_i} * \sqrt{67}^{-x} = \sqrt{67}^{0} = f(0) = f(\lim_{i\to\infty} y_i).$

$Seien\ \ y_i, x_i\ eine\ beliebige\ rationale\ Folge\ \ mit\ \lim_{i\to\infty} y_i = y$

$$\lim_{i\to\infty} f(y_i) = \lim_{i\to\infty} f(y + (y_i - y)) = \lim_{i\to\infty} f(y) \lim_{i\to\infty} f(y_i - y) = f(y) = f\left(\lim_{i\to\infty} y_i\right)$$

Also ist f stetig und erfüllt alle Bedingungen.

2.3 Funktionalgleichung III (nicht erfüllbar)

$Finde\ alle\ monotonen\ Funktionen\ f: \mathbb{R} \to \mathbb{R}\ mit\ f(x + f(y)) = f(x) + y^2$

$I. y \to 0: f(x + f(0)) = f(x)$

$Angenommen\ f(0) = c \neq 0.$ Dann gilt f(x + c) = f(x) = k.

$Und\ somit\ f(x) = f(x + c) = f\big((x + c) + c\big) = f(x + t * c) für\ alle\ natürlichen\ t.$

Aufgrund der Monotonie gilt somit $f(x) = k$ für alle reellen x.

$Dann\ entsteht\ jedoch\ der\ Widerspruch\ f(x + f(y)) = k \neq k + y^2\ für\ y \neq 0.$

Also gilt $f(0) = 0$

II. $x \to 0: f\big(f(y)\big) = y^2 = f\big(f(-y)\big)$
Da f monoton ist, $gilt\ dies\ dann\ jedoch\ auch\ für\ k\ mit: -y < k <$
$y\ was\ sich\ mit\ der\ Voraussetzung\ f\big(f(k)\big) = k^2\ widerspricht.$ Also ist f nicht erfüllbar.

2.4 Funktionalgleichung IV (Komposition: exponentiell und quadratisch)

$Finde\ alle\ stetigen\ Funktion\ f: \mathbb{R} \to \mathbb{R}\ mit\ 1. f(x + y)f(x - y) = f(x)^2 f(y)^2\ und\ 2. f(1) = e$

$x = y = 0: f(0)^2 = f(0)^4. Also\ f(0) = 1$

$x \to \frac{x}{2}, y \to \frac{x}{2}: f(x) = f\left(\frac{x}{2}\right)^4$

$\ln\big(f(x)\big) = \ln\left(f\left(\frac{x}{2}\right)^4\right) = 4\ln\left(\frac{x}{2}\right)$

$Sei\ g(x) = \ln\big(f(x)\big), dann\ gilt\ e^{g(x)} = f(x).\ Es\ gilt\ g(1) = 1\ und\ g(0) = 0$

$Weiter\ gilt\ g(1) = 4^n g\left(\frac{1}{2^n}\right) = 1.$

Sei x = $\frac{1}{2^n}, wobei\ n\ reell. Dann\ gilt\ g(x) = \frac{1}{4^n} = \left(\frac{1}{2^n}\right)^2 = x^2$

$Da\ f(x) = e^{x^2} differenzierbar\ und\ somit\ stetig\ ist, erf$üllt f $alle\ Bedingungen.$

2.5 Funktionalgleichung V (D´Alembert trigonometrisch)

$Finde\ alle\ stetigen\ Funktionen\ f: \mathbb{R}+\rightarrow \mathbb{R}\ mit\ 1. f(x+y) + f(x-y) = 2f(x)f(y),$

$2. f''(0) < 0\ und\ 3. f(\pi) = -1$

$x = 0: 2\ f(x) = 2f(x)f(0), also\ f(0) = 1$

$$\frac{\partial f(x+y) + f(x-y)}{\partial y} = f'(x+y) - f'(x-y) = \frac{\partial 2f(x)f(y)}{\partial y} = 2f'(x)f(x)$$

$x = y = 0:\ 0 = f'(0) - f'(0) = 2f'(0)f(0)\ .Also\ ist\ f'(0) = 0$

$$\frac{\partial f'(x+y) + f'(x-y)}{\partial y} = f''(x+y) - f''(x-y) = \frac{\partial 2f'(y)f(x)}{\partial y} = 2f''(y)f(x)$$

$y = 0\ und\ f''(0) = k < 0:\quad 2f''(x) = 2f''(0)f(x). Also\ f''(x) + |k| * f(x) = 0$

$Die\ charakteristische\ Gleichung\ \lambda^2 + |k| = 0\ hat\ die\ komplexe\ L$ösung$: \sqrt{|k|}i$

$Also\ entspricht\ f(x) = c_1 \sin\left(\sqrt{|k|}x\right) + c_2 \cos\left(\sqrt{|k|}x\right)$

$Aus\ f(0) = 1, sowie\ der\ 2. und\ 3. Bedingung\ folgt\ f(x) = \cos(x)$

3 Loss Funktionen

3.1 Gradienten des MLP

$b_i(x) = c_i * x + k_i \; mit \; \frac{\partial b_i}{\partial c_i} = x$

$r(x)=\begin{cases} x, falls\ x > 0 \\ 0, sonst \end{cases} mit \; \frac{\partial r(b_i)}{\partial c_i}=\begin{cases} x, falls\ b_i(x) > 0 \\ 0, sonst \end{cases}$

$\tanh(x), mit \; \frac{\partial \tanh(b_i)}{\partial c_i} = (1 - \tanh(cx + k)^2)x$

$Sei\ unser\ MLP\ vereinfacht\ dargestellt\ als\ f(x)$

$$= \sum_{i=1}^{n} c'_i r\big(b_i(x)\big) + k'_i, beziehungsweise$$

$$f(x) = \sum_{i=1}^{n} c'_i tanh(b_i(x)) + k'_i$$

$mit \; \frac{\partial f}{\partial c_i} = c'_i * \frac{\partial r(b_i(x))}{\partial c_i}$ und $\frac{\partial f}{\partial c'_i} = r(b_i(x))$, beziehungsweise $\frac{\partial f}{\partial c_i} = c'_i * \frac{\partial tanh(b_i(x))}{\partial c_i}$ und $\frac{\partial f}{\partial c'_i} = tanh(b_i(x))$,

3.2 Loss Funktionen

Für die erste Funktionalgleichung ist eine Verlustfunktion gesucht, mit der die Additivität, das iterative Verhalten und Positivität erlernt werden kann:

$$Loss_1(x, y) = (f(x + y) - \big(f(x) + f(y)\big)^2 + \big(f\big(f(x)\big) - 5x\big)^2 + \lambda * relu(-f(x))$$

Mit $\frac{\partial Loss_1}{\partial \theta} = 2 * (f(x + y) - \big(f(x) + f(y)\big) * \left(\frac{\partial f(x+y)}{\partial \theta} - \frac{\partial f(x)}{\delta \theta} - \frac{\partial f(y)}{\partial \theta}\right) + 2 * \big(f\big(f(x)\big) - 5x\big) * \frac{\partial f(f(x,\theta),\theta)}{\partial \theta} + 2 * \lambda * 1_{f(x)>0} * \frac{\partial f(x)}{\partial \theta}$

Wobei $\theta\ ein\ Parameter\ von\ f\ ist.$

Für die zweite Funktionalgleichung ist eine Verlustfunktion gesucht, mit der die Exponentialfunktion mit einer eindeutigen Basis erlernt wird:

$$Loss_2(x,y) = (f(x+y) - f(x) * f(y))^2 + \lambda * (f(2) - 67)^2 + \lambda * (f(0) - 1)^2$$

$$Mit\ \frac{\partial Loss_2}{\partial \theta} = 2 * \big(f(x+y) - f(x) * f(y)\big) * (\frac{\partial f(x+y)}{\partial \theta} - \frac{\partial f(y)}{\partial \theta} * f(x) - \frac{\partial f(x)}{\partial \theta} * f(y) + 2 * \lambda * (f(2) - 67) * \frac{\partial f(2)}{\partial \theta} + 2 * \lambda * (f(0) - 1) * \frac{\partial f(0)}{\partial \theta}$$

Für dritte Funktionalgleichung soll mittels der Verlustfunktion monotones Verhalten, sowie die nicht erfüllbare Gleichung erlernt werden.

$$Loss_3(x,y) = (f\big(x + f(y)\big) - (f(x) * + y^2))^2 + \lambda * relu(f(x) - f(x + epsilon)$$

$$Mit\ \frac{\partial Loss_3}{\partial \theta} = 2 * \big(f\big(x + f(y)\big) - (f(x) * + y^2)\ \big) * (\frac{\partial f(x+f(y,\theta),\theta)}{\partial \theta} - \frac{\partial f(x)}{\partial \theta}) + \lambda * 1_{f(x)>0} * \frac{\partial f(x) - f(x+epsilon)}{\partial \theta}$$

Für die vierte Funktionalgleichung soll mittels der Verlustfunktion die allgemeine FG, sowie die eindeutige Basis e erlernt werden.

$$Loss_4(x,y) = (f(x+y) - f(x)^2 * f(y)^2)^2 + \lambda * (f(1) - e)$$

$$Mit\ \frac{\partial Loss_4}{\partial \theta} = 2 * (f(x+y) - f(x)^2 * f(y)^2) * (\frac{\partial f(x+y)}{\partial \theta} - (\frac{\partial f(x) f(x)}{\partial \theta} f(y)^2 + f(x)^2 \frac{\partial f(y) f(y)}{\partial \theta}) \lambda * 2_{f(x)>0} * (f(1) - e) \frac{\partial f(1)}{\partial \theta}$$

Für die fünfte Funktionalgleichung soll mittels der Verlustfunktion die allgemeine FG von D´Alembert, sowie die Zusatzbedingungen, die eine eindeutige Lösung erzwingen, erlernt werden.

$$Loss_5(x,y) = (\big(f(x+y) + f(x-y)\big) - 2f(x)f(y))^2 + relu\big(f''(0)\big) + \lambda * (f(\pi) + 1)^2$$

Mit $$\frac{\partial Loss_5}{\partial \theta} = 2\Big(\big(f(x+y) + f(x-y)\big) - 2f(x)f(y)\Big) * (\frac{\partial f(x+y) + f(x-y)}{\delta \theta} - 2 * (\frac{\partial f(x)}{\partial \theta} f(y) + f(x) \frac{\partial f(y)}{\partial \theta}) + 1_{f''(x)>0} \frac{\partial f''(0)}{\partial \theta} + 2\lambda (f(\pi) + 1)(\frac{\partial f(\pi)}{\partial \theta})$$

4 Modellarchitekturen

4.1 Architekturen

Die Modelle weisen folgende Multi Layer Perceptron Architektur auf:

```python
def MLPRelu (n : int):  1 usage
    """Simple MLP model"""
    MLP = nn.Sequential(
        nn.Linear( in_features: 1, n, bias=True),
        nn.ReLU(),
        nn.Linear(n,n, bias=True),
        nn.ReLU(),
        nn.Linear(n,  out_features: 1, bias=True)
    )
    return MLP

def MLPTanh (n : int):  1 usage
    """Simple MLP model"""
    MLP = nn.Sequential(
        nn.Linear( in_features: 1, n, bias=True),
        nn.Tanh(),
        nn.Linear(n,n, bias=True),
        nn.Tanh(),
        nn.Linear(n,  out_features: 1, bias=True)
    )
    return MLP
```

Abbildung 1: Modellarchitekturen in Pytorch

Sie besitzen somit $n^2 + 4n + 1$ erlernbare Parameter und sind kompakt genug, um eine Funktionalgleichung in verhältnismäßig kurzer Zeit numerisch zu lösen. Die Zahl der Neuronen pro Hidden Layer (131), die Dämpfungsfaktoren, sowie die Zahl der Trainingsepochen (170) wurden mittels Simulated Annealing (Kirkpatrick, Gelatt and Vecchi, 1983) ermittelt. Der Suchraum entsprach:

Hidden Layer - Zahl der Neuronen: 50 bis 400

Epochen : 50 bis 300

Dämpfungsfaktoren: 0.1 bis 2.2

```python
def optimalHyperParameters( epochs=40, temperature = 1000, decay=0.93):  1 usage
    """searches for better performing hyperparameters using simulated annealing algorithm"""
    # hiddenLayerdims, epochs, array of
    currentConfig = [100, 200, [1.0, 1.0, 1.0, 1.0]]
    optimalConfig = currentConfig

    minLoss = getLoss(currentConfig)
    saveConfigToFile(currentConfig, filename: "SALogs.txt", minLoss)


    for e in range(epochs):
        print("epoch:", e)
        temperature *= decay
        newConfig = getNewConfig(currentConfig)

        newLoss = getLoss(newConfig)

        if newLoss <  minLoss:
            currentConfig = newConfig

            optimalConfig = newConfig
            saveConfigToFile(optimalConfig, filename: "SALogs.txt",newLoss)
            minLoss = newLoss
        elif np.exp(-((newLoss-minLoss)/temperature)) > np.random.rand():
            currentConfig = newConfig


    return optimalConfig
```

Abbildung 2: Simulated Annealing Implementierung

Das Training erfolgt in Batches mittels des Adam Optimizer, der sich wegen seiner adaptiven Lernrate (Kingma and Ba, 2015) anbietet:

```python
def trainedMLP (model, epochs, dampingFactor, functionalEquation, trainingData):
    """train model indirectly on functional equation"""

    optimizer = optim.Adam(model.parameters(), lr=0.001)
    dataloader = DataLoader(trainingData, batch_size: 1000, shuffle=True)

    if functionalEquation == 'linear':
      for i in tqdm(range (epochs)):
          for data in dataloader:
            inputX = data[0]
            inputY = data[1]
            fX = model(inputX)
            fY = model(inputY)
            fXPlusY = model(inputX + inputY)
            ffx = model(fX)

            loss = lossOne(fXPlusY, fX, fY, ffx, inputX, dampingFactor)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

Abbildung 3: Training in Pytorch (Beispiel)

4.2 Alternative Modelle

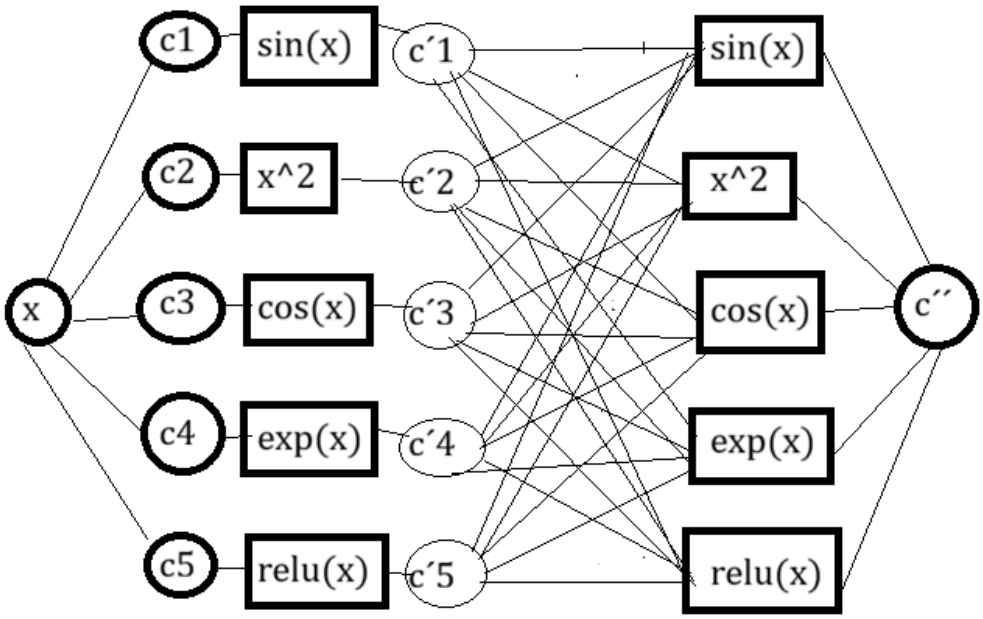

Abbildung 4:Analytisches Repräsentationsmodell

Ein Modell wie das obige könnte prinzipiell alle durch die Funktionalgleichungen ausgedrückten Funktionen mit gerade einmal 5 + 5 + 25 Parametern perfekt erlernen und würde zugleich eine kompakte symbolische Repräsentation ermöglichen. Jedoch ist es numerisch problematisch, da die stark divergierenden Wachstumseigenschaften der verschiedenen Aktivierungsfunktionen ein gradientenbasiertes Optimierungsverfahren, wie das des Gradientenabstiegs in Kombination mit Backpropagation (Goodfellow, Bengio and Courville, 2016) so gut wie unmöglich machen. Approximative Netze sind hier weitaus überlegen.

5 Auswertung

Für alle Auswertungen gilt:

$$\text{validationLoss} = |\text{mlp(x)}/\max(\text{mlp(x)}) - \text{f(x)}/(\max(\text{f(x)})| \,/\, \text{N}$$

wobei $f(x)$ die zu erlernende Funktion ist, x ein Vektor mit Testinputs aus derselben Domäne wie die Trainingsdaten und N die Zahl der Einträge in x.

5.1 Experiment 1

Zahl der Trainingstupel (x,y): 50000, Trainingsdauer CPU: 01:50 bis 02:20 pro MLP

Cauchy Linear (Domäne: [0, 1000])

Relu MLP Validation Loss: 0.0000*2775657412712*
Tanh MLP Validation Loss: 0.5240769386291504

Cauchy Exponentiell (Domäne: [-1, 1])

Relu MLP Validation Loss: 0.09429547935724258
Tanh MLP Validation Loss: *0.04799851402640343*

Nicht erfüllbare FG (Domäne: [-100, 100])

Relu MLP FE Loss: :*1216141824.0*

Tanh MLP FE Loss: *22340964352.0*

Komposition Exponential *(Domäne: [-1, 1])*

Relu MLP Validation Loss: *0.10941815376281738*
Tanh MLP Validation Loss: *0.32230302691459656*

D´Alembert Trigonometrisch(Domäne: [-π, π])

Relu MLP Validation Loss: *0.2448619306087494*
Tanh MLP Validation Loss: *0.21657611429691315*

5.2 Experiment 2

Zahl der Trainingstupel (x,y): 100000, Trainingsdauer CPU: 02:50 bis 04:00 pro MLP

Cauchy Linear (Domäne: [0, 1000])

Relu MLP Validation Loss: 0.00008243082265835
Tanh MLP Validation Loss: 0.48658111691474915

Cauchy Exponential (Domäne: [-1, 1])

Relu MLP Validation Loss: 0.07350177317857742

Tanh MLP Validation Loss: *0.0734217755157622*

Nicht erfüllbare FG (Domäne: [-100, 100])

Relu MLP FE Loss: 11783735.0

Tanh MLP FE Loss: *162510581.0*

Komposition Exponential *(Domäne: [-1, 1])*

Relu MLP Validation Loss: 0.103350929915905

Tanh MLP Validation Loss: 0.11845000088214874

D´Alembert (Domäne: $[-\pi, \pi]$)

Relu MLP Validation Loss: *0.24869756400585175*

Tanh MLP Validation Loss: *0.22954165935516357*

6 Fazit

Die vorliegende Arbeit zeigt, dass MLP grundsätzlich geeignet sind, stetige Lösungen ausgewählter Funktionalgleichungen numerisch zu approximieren. Durch die gezielte Konstruktion der Verlustfunktionen konnten strukturelle Eigenschaften der Funktionalgleichungen in den Optimierungsprozess eingebettet werden. Insbesondere die Ergebnisse zur additiven Cauchy-Gleichung belegen, dass ein ReLU-basiertes MLP eine sehr geringe Validierungsabweichung erreichen kann, während Tanh-Netze in diesem Szenario deutlich schlechter konvergierten. Dies ist nicht weiter verwunderlich da eine ReLU Aktivierung bei positiven Eingangswerten linear ist, während ein tanh Aktivierung strikt nicht linear ist.

Für exponentielle Zusammenhänge zeigte Tanh teilweise Vorteile gegenüber ReLU, was auf die glattere Nichtlinearität und bessere Modellierung symmetrischer Bereiche hindeutet. Bei der nicht erfüllbaren Funktionalgleichung divergierten beide Modelle erwartungsgemäß, wobei die stark ansteigenden Fehlerwerte als Indikator für die Inkonsistenz der Nebenbedingungen interpretiert werden können.

Die Wahl der Hyperparameter mittels Simulated Annealing erwies sich als praktikabler Ansatz, um die Zahl der Neuronen pro Schicht, Dämpfungsfaktoren und Trainingsdauer auszubalancieren.

Zusammenfassend lässt sich festhalten, dass neuronale Netze ein flexibles Werkzeug zur numerischen Behandlung von Funktionalgleichungen darstellen, jedoch keine Garantie für eindeutige oder interpretierbare Lösungen bieten. Die Qualität der Resultate hängt entscheidend von der Problemformulierung, der Loss-Definition und der Modellarchitektur ab.

Literaturverzeichnis

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press

Hornik, K., Stinchcombe, M. and White, H. (1989) ‘Multilayer feedforward networks are universal approximators’, *Neural Networks*, 2(5), S. 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Kingma, D.P. and Ba, J. (2015) ‘Adam: A Method for Stochastic Optimization’. arXiv:1412.6980

Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) ‘Optimization by simulated annealing’, *Science*, 220(4598), pp. 671–680. https://doi.org/10.1126/science.220.4598.671