Verwendete Python Bibliotheken:

- Vorraussetzung: Pip Paketmanager
- Paketinstallationen können durch Ausführen des Notebook-Code-Blocks durchgeführt werden

In [ ]:
```
! pip install pylatex==1.4.1                                                    #
! pip install ipykernel=6.17.1                                                  #
! pip install ipython==8.7.0                                                    #
! pip install jupyter_client==7.4.7                                             #
! pip install jupyter_core==5.1.0                                               #
! pip install latexcodec2.0.1                                                   #
! pip install matplotlib-inline==0.1.6                                          #
! pip install plotly==5.11.0                                                    #
! pip install Sphinx (                                                          #
!   applehelp==1.0.2,
!   bibtex==2.5.0,
!   devhelp==1.0.2,
!   htmlhelp==2.0.0,
!   jsmath==1.0.2,
!   qthelp==1.0.3,
!   serializinghtml==1.1.5
! )
! pip install numpy==1.22.0                                                      #
```

```
ERROR: Invalid requirement: '#'
ERROR: Invalid requirement: 'ipykernel=6.17.1'
Hint: = is not a valid operator. Did you mean == ?
```

Verwendete VSCode Extensions

In [ ]:
```
# IntelliCode                    v.1.2.29                by Microsoft
# Jupyter                        v2022.11.1003412109     by Microsoft
# Jupyter Cell Tags              v0.1.6                  by Microsoft
# Jupyter Keymap                 v1.0.0                  by Microsoft
# Jupyter Notebook Renderers     v1.0.12                 by Microsoft
# Jupyter slide Show             v0.1.5                  by Microsoft
# Pylance                        v2022.12.20             by Microsoft
# Python                         v2022.20.0              by Microsoft
```

Stochastische Optimierung: ADAM

Inhaltsverzeichnis

- Optimierer und ihre Verwendungszwecke
  - Definition der Verlustfunktion
  - Definition Modell
  - Wie können Modelle optimiert werden?
- Adam Optimierer
- Beispiele
- Einsatzbereich
- Limitierungen

- Alternativen und Abgrenzung
- Fazit
- Quellen
  - Eingesetzte Quellen
  - Weiterführende Quellen

Abkürzungsverzeichnis

Abkürzung	Bedeutung
p	Wahrscheinlichkeit
CSAM	Child Sexual Abuse Material
MQA	Mittlere Quadratische Abweichung
SGD	Stochatische Gradientenabstieg

Formelgrößen und Einheiten

Abkürzung	Einheit	Bezeichnung
α		Schrittgröße
β		Gewichtung neuer Werte gegenüber alter Werte
β_1		Mittelwert
β_2		zentrierte Varianz
f		Funktion mit dem Parameter θ
m_0		Mittelwert des Gradienten
v_0		unzentrierte Varianz des Gradienten
δ		Gradient
t	Zeiteinheit (beliebige Größe)	Zeit
ϵ		Konstante

Optimierer und ihre Verwendungszwecke

Bevor die Funktionsweise von Optimierern im Detail erläutert wird, werden zwei mit Optimierern zusammenhänge Begriffe, Verlustfunktion und Machine Learning Modell, definiert.

Definition der Verlustfunktion

Die Verlustfunktion ordnet in einem Entscheidungsraum (Ω, Σ) Bereichsschätzungen und Punktschätzungen den Schaden zu, der durch eine vom wahren Parameter abweichende Entscheidung verursacht wird. Die

Verlustfunktion wird innerhalb eines statistischen Models $(X, \mathcal{A}, (P_\vartheta)_{\vartheta \in \Theta})$ verwendet und ist wie folgt aufgebaut: $L : \Theta \times \Omega \to [0, +\infty]$. Dabei gilt, das die Funktion $L(\vartheta, \cdot)\Sigma - \mathcal{B}([0, +\infty])$ für jedes fixierte $\vartheta \in \Theta$ messbar ist. Mehr Informationen zur Verlustfunktionen können Rüschendorf, 2014 entnommen werden.

Verlustfunktionen können auf einzelne Anwendungsfälle angepasst werden. Die folgene Beispielverlustfunktion berechnet die Differenz zwischen vorhergesagten Werten und den tatsächlichen Werten $(Vorhersage - Realwert = Differenz)$. Je höher die Differenz, desto höher ist der verursachte Schaden einer Bereichs- oder Punktschätzung. Wenn zum Beispiel der geschätzte Wert für die monatlichen Wohnungskosten in Stuttgart 1500 beträgt, jedoch in der Realität die Kosten bei 1000 liegen, dann gibt die Verlustfunktionen eine Differenz von 500 aus. Wenn die Vorhersage unter dem Realwert liegt, generiert die Verlustfunktion ebenfalls eine Differenz Datarobot, 2018.

| Vorhersage | Realwerte | Differenz | |----------|---------- -|----------| | 1500 | 1000 | 500 | | 1000 | 1000 | 0 | | 750 | 1000 | -250 |

Die Verlustfunktion könnte dahingehend angepasst werden, dass positive Abweichungen $(Vorhersage > Realwert)$ einen höheren Schaden verursachen, als negative Abweichungen $(Vorhersage < Realwert)$. Um den Gesamtverlust innerhalb eines Datensatzes zu berechnen, kann die quadratische Abweichung $((Vorhersage - Realwert)^2 = Quadratfehler)$ zwischen allen Vorhersagen und Realwerten berechnet werden. Alle Quadratfehler werden summiert und deren Mittlere quadratische Abweichung (MQA) wird berechnet Datarobot, 2018.

$$\text{MQA} = \frac{1}{n} \sum_{i=1}^{n} (Vorhersage_i - Realwert_i)^2$$

Eine andere Möglichkeit den Verlust der Verlustfunktion zu berechnen ist die Likelihood Funktion. Die Likelihood Funktion erhält als Eingabe die vorhergesagte Wahrscheinlichkeit (p) des Eintretens eines Events und multipliziert die Eingabewerte miteinander. Wenn zum Beispiel folgende Vorhersagen $([0.2, 0.35, 0.1, 0.87])$ in Kombination mit folgenden Wahrheitswerten $([1, 0, 0, 1])$ getroffen wurden, berechnet sich der Wert der Likelihood Funktion wie folgt:

$$0.2 * (1 - 0.35) * (1 - 0.1) * 0.87 = 0.102$$

Für alle falschen Werte wird die Wahrscheinlichkeit über $1 - p$ berechnet Datarobot, 2018.

Definition Modell

Ein Machine Learning Modell ist definiert als Datei, die bestimmte Muster erkennt QuinnRadich et al., 2022. Die Musterkennung des Modells wird durch Training mit Daten verbessert. Durch das Training entsteht ein Vorhersagemodell, welches mit einer hohen Wahrscheinlichkeit Vorhersagen anhand von Eingabedaten treffen kann Murdoch et al, 2019, S. 2.

Wie können Modelle optimiert werden?

Mit Hilfe der Verlustfunktion können Modelle anhand ihrer Gewichte optimiert werden. Dieser Prozess kann manuell durchgeführt werden. Dabei werden die Gewichte so angepasst, dass die Verlustfunktion minimiert wird. Ein Optimierer automatisiert diesen Prozess. Er setzt die Ausgabe der Verlustfunktion und die Eingabe der Gewichte in Relation und optimiert die Gewichte so, dass die Verlustfunktion minimiert wird Datarobot, 2018.

Ein beispielhafter Optimierer ist der Gradientenabstieg. Der Gradientenabstieg optimiert Modelle in zwei Schritten Datarobot, 2018:

1. Berechne für jedes Gewicht welchen Einfluss kleine Änderungen auf die Verlustfunktion haben. Dieser Einfluss auf die Verlustfunktion wird Gradient genannt. Ein Gradient ist ein Spaltenvektor, der die partiellen Ableitungen von f nach den Spaltenvektoren (also den Gewichten) enthält Studyflix, o. J.
2. Modifiziere die Gewichte anhand der Gradienten, sodass die Verlustfunktion kleiner wird.

Die Schritte eins und zwei werden so lange wiederholt, bis die Verlustfunktionen einen festgelegten minmalen Wert erreicht hat. Die Geschwindigkeit mit dem der Gradientenabstieg operiert, kann über die Lernrate (alternativ auch Schrittgröße genannt) angepasst werden. Die Lernrate bestimmt die Größe der Änderungen, die der Gradientenabstieg an den Gewichten vornimmt. Diese wird in Kommazahlen angegeben und hat meistens Werte von 0.0001 bis 0.001. Die Gradienten werden mit der Lernrate multipliziert. Eine zu kleine Lernrate kann dazu führen, das der Gradientenabstieg Probleme mit lokalen Minima hat. In diesem Fall geht der Optimierer davon aus, dass er den minimalsten Punkt erreicht hat, jedoch handelt es sich bei dem Punkt nur um ein lokales Minimum Datarobot, 2018.

Ein weiteres Problem für Optimierer kann Overfitting darstellen. Beim Overfitting wird das Model zu sehr auf die Daten mit denen es trainiert wurde angepasst. Dadurch kann das Modell mit den Trainingsdaten nahezu perfekt arbeiten, hat jedoch Probleme mit realen Daten. Overfitting tritt auf, wenn einzelne Gewichte einen zu hohen Einfluss auf das Model haben und dadurch das Ergebnis zu stark modifizieren. Optimierer können Overfitting durch das Hinzufügen von einem zusätzlichen Strafparameter zur Verlustfunktion vermeiden. Der Strafparameter bestraft Optimierer, wenn sie zu große Werte für Gewichte nehmen, auch wenn dadurch die Verlustfunktion kleiner wird. Dadurch verändern Optimierer mehrere Gewichte in kleinen Schritten, anstatt einzelne Gewichte in großen Schritten Datarobot, 2018.

Im Folgenden wird ein komplexerer Optimierer, der Adam Optimierer, beschrieben.

Adam Optimierer

Der Adam Optimierer ist ein Algorithmus für stochastische, gradientenbasierte Optimierung erster Ordnung von stochastischen Zielfunktionen Diederik et al., 2017. 2014 wurde dieser auf der ICLR Konferenz für Deep-Learning-Forscher von Jimmy Ba und Diederik Kingma vorgestellt Diederik et al., 2017. Dieser zeichnet sich durch folgende Punkte aus:

- einfache Implementierung
- rechnerische Effizienz
- geringer Speicherbedarf
- invariant zu diagonalen Neuskalierungen der Steigungen
- gut geeignet für Probleme in Bezug auf Daten oder Parameter

Dieser Optimierer ist weit verbreitet und wird beim Training von neuronalen Netzen eingesetzt Introduction to Optimizers, Adam.

Mit diesem Optimierer, beziehungsweise dieser Methode, können individuelle adaptive Lernraten für unterschiedliche Parameter aus Schätzungen der ersten und zweiten Momente der Gradienten berechnet werden. Abgeleitet ist der Name Adam von 'adaptive moment estimation'. Diese Methode soll die Vorteile der beiden Methodiken AdaGrad Duchi et al., 2011 sowie RMSProp Tielemann & Hinton, 2012 kombinieren.

So gehören zu den Vorteilen von ADAM, dass

- die Größenordnungen der Parameteraktualisierungen invariant gegenüber der Skalierung der Gradienten sind,
- die Schrittweiten näherungsweise durch den Hyperparameter der Schrittweite begrenzt sind,
- kein stationäres Ziel erforderlich ist,
- Adam mit spärlichen Gradienten arbeiten kann und
- Adam eine Form von Schrittgrößen-Annealing durchführt.

Der Algorithmus ist wie folgt aufgebaut:
Nutzer müssen eine Schrittweite α definieren. Darüber hinaus müssen β_1 ud β_2 bestimmt werden. Diese bestimmen die Gewichtung neuer Werte gegenüber älteren Werten (Momentum Optimizerin Deep Learning, 2021, Exponentially Weighted Moving Average (2:53)). Darüber hinaus definieren Nutzer eine Funktion mit dem Parameter θ, sowie den initialen Parameter Vektor θ_0.

Beim Durchführen von Adam werden folgende Schritte durchgeführt:

- m_0 = Mittelwert des Gradienten wird inititalisiert
- v_0 = Unzentrierte Varianz des Gradienten wird initialisiert
- t = Der erste Zeitpunkt wird initialisiert

Der folgende Codeblock wird solange wiederholt, bis θ_t konvergiert:

- t <- $t+1$ = Erhöhe den Zeitpunkt um eins
- g_t <- $\nabla_0\ f_t\ (\theta_{t-1})$ = Erhalte alle Gradienten (∇) an der Stelle des aktuellen Zeitpunktes
- m_t <- $\beta_1 * m_{t-1} + (1-\beta_1) * g_t$ = Aktualisiere die Schätzung der Mittelwerte des Gradienten
- v_t <- $\beta_2 * m_{t-1} + (1-\beta_2) * g_t^2$ = Aktualisiere die Schätzung der unzentrerten Varianz des Gradienten
- $\hat{m}_t$ <- $m_t/(1-\beta_1^t)$ = Berechne die fehlerkorrigierte Schätzung der Mittelwerte des Gradienten
- $\hat{v}_t$ <- $v_t/(1-\beta_2^t)$ = Berechne die fehlerkorrigierte Schätzung der unzentrierten Varianz des Gradienten

Sobald θ_t konvergiert, werden die berechneten Parameter ausgegeben.

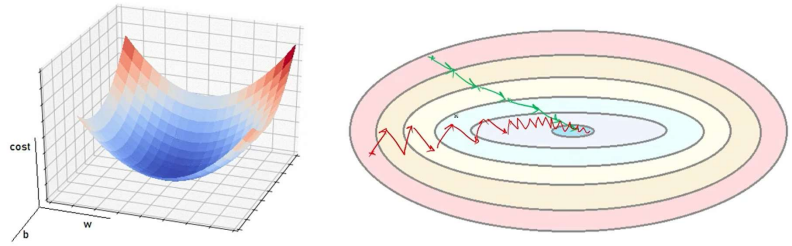
Der Adam Optimierer ist eine Kombination des Momentum Optimierers sowie des RMSprop Optimierers. Momentum und RMSprop bestehen aus aus vier mathematischen Formeln. RMSprop und Momentum basieren auf dem Gradientenabstieg. Dieser wurde im Kapitel "Wie können Modelle optimiert werden?" erklärt.

Momentum (Momentor Optimizer in Deep Learning, 2021):

1. $V_{dW} = \beta * V_{dW_{prev}} + (1-\beta) * dW$
2. $V_{dB} = \beta * V_{dB_{prev}} + (1-\beta) * dB$
3. $W = W - \alpha * V_{dW}$
4. $B = B - \alpha * V_{dB}$

Die Vorgabe ist, dass zwei Gewichts-Parameter (weight-parameter) eingesetzt werden, um das lokale Minimum zu erreichen. Da zwei Gewichts-Parameter vorliegen, bewegt man sich im dreidimensionalen Raum.

In [2]:
```
from IPython.display import display, Image
display(Image(filename="Adam_opt.png", height=400, width=400))
```



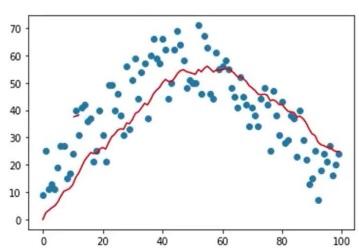
Das Ziel dieses Algorithmus (dargestellt in rot) ist es, die horizontale Bewegung zu verstärken und gleichzeitig die vertikale Bewegung zu reduzieren, um auf diesem Weg das lokale Minimum zu finden. Dazu wird folgende Formel verwendet, die eine allgemeinere Form der Formeln 1. und 2. darstellt:
$V_t = \beta * V_{t-1} + (1-\beta) * \theta_t$.
Mit jedem Datenpunkt wird das (Durchschnitts-)Gewicht neu berechnet. Dabei muss bei jeder Neuberechnung von V abhängig von der Zeit immer das Gewicht des vorherigen Datenpunktes bzw. Schrittes mit berücksichtigt werden. β wird dabei zwischen Null und Eins gewählt ($0 < \beta < 1$).

Diese Formel kann für die horizontale Bewegung, wie auch für die vertikale Bewegung eingesetzt werden, sollte aber anhand der Variablenbezeichnungen leicht angepasst werden. Die Formeln 3. und 4. stellen die eigentliche Bewegung dar, die abhängig von der jeweiligen Gewichtung aus 1. oder 2. ist.

Visualisiert kann dies folgendermaßen dargestellt werden (dabei zeigt die rote Linie das Durchschnittsgewicht an jeder Stelle der Punkte an).

In [3]:
```
from IPython.display import display, Image
display(Image(filename="Adam_v.png", height=400, width=400))
```

$V_0 = 0$
$V_1 = \beta * V_0 + (1-\beta) * \theta_1$
$V_2 = \beta * V_1 + (1-\beta) * \theta_2$
$V_3 = \beta * V_2 + (1-\beta) * \theta_3$
$V_4 = \beta * V_3 + (1-\beta) * \theta_4$
$V_5 = \beta * V_4 + (1-\beta) * \theta_5$
$V_6 = \beta * V_5 + (1-\beta) * \theta_6$
$V_7 = \beta * V_6 + (1-\beta) * \theta_7$
$V_8 = \beta * V_7 + (1-\beta) * \theta_8$
$V_9 = \beta * V_8 + (1-\beta) * \theta_9$



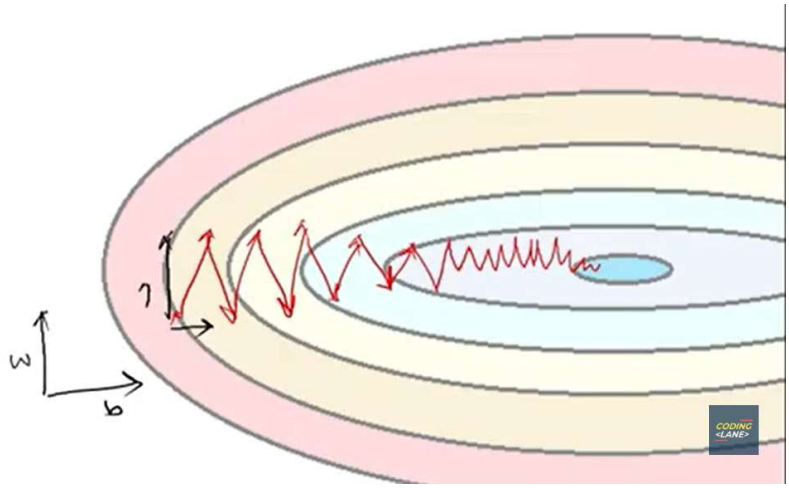
RMSprop (RMSprop Optimizer Explained in Detail, 2021):

1. $S_{dW} = \beta * S_{dW_{prev}} + (1-\beta) * (dW)^2$
2. $S_{dB} = \beta * S_{dB_{prev}} + (1-\beta) * (dB)^2$
3. $W = W - \alpha * (\frac{dW}{\sqrt{S_{dW}+\epsilon}})$
4. $B = B - \alpha * (\frac{dB}{\sqrt{S_{dB}+\epsilon}})$

RMSprop verfolgt das gleiche Ziel, nur mit einer geringfügig abgewandelten Methodik. Bei 5. und 6. wird der dW und dB Wert quadiert. Zur Unterscheidung wird der berechnete Faktor mit S abgekürzt und nicht wie bei Momentum mit V. Wächst somit der Faktor, z.B. S_{dW} so wird die vertikale Bewegung reduziert. Das

gilt invertiert auch für die horizontale Bewegung. Umso kleiner der Faktor, desto größer ist die horizontale Bewegung. Damit kann auch schneller das lokale Minimum erreicht werden. Grund dafür ist, dass der Faktor für die Bewegungsberechnung im Nenner steht und somit für einen kleinen Wert sorgt. Aus diesem Optimierungsvorgehen ergibt sich auch die Bezeichnung von RMSprop - Root Mean Square propagation.

In [4]:
```
from IPython.display import display, Image
display(Image(filename="RMSprop_wb.png", height=400, width=400))
```



Werden diese beiden Optimierer in der Folge kombiniert, so ergibt sich für die horizontale Bewegung folgende Gleichung:

$$W = W - \alpha * (\frac{V_{dW}}{\sqrt{S_{dW} + \epsilon}})$$

Für die vertikale Bewegung ergibt sich diese Gleichung:

$$B = B - \alpha * (\frac{V_{dB}}{\sqrt{S_{dB} + \epsilon}})$$

Beispiel

Im Folgenden wird eine Beispielimplementierung in Python beschrieben. Die Implementierung basiert auf einem Beispiel von Enoch Kan, 2020. Das Beispiel verwendet die Biblliotheken numpy und math.

Um den Adam Algorithmus zu initialisieren müssen einige Variablen definiert werden:

- alpha (α) = Schrittgröße
- beta1 (β_1) und beta2 (β_2) = Gewichtung neuerer Werte gegenüber älteren Werten. beta1 fasst den Mittelwert und beta2 die unzentrierte Varianz zusammen. Bei der unzentrierten Varianz handelt es sich um den reinen Varianzwert ohne Abzug des Mittelwerts.
- epsilon (ϵ) = Konstante, damit beim Updaten von nicht durch null geteilt wird.

In jeder Iteration des Algorithmus werden folgende Schritte durchgeführt: Im Ersten Schritt werden die Gewichte und der Bias von β_1 und β_2 berechnet. Dazu werden m_t (der Mittelwert) und v_t (die Varianz) über die Formeln
$m_t = \beta_t * m_{t-1} + (1 - \beta_1) * g_t$ und $v_t = \beta_t * v_{t-1} + (1 - \beta_2) * g_t^2$ berechnet.

Die gleitenden Mittelwerte werden über die Parameter β_1 und β_2 und den Gradienten g_t berechnet. Dieser Schritt wird Bias Correction genannt und wird über folgende Formeln $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ und $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ durchgeführt. Die Gewichte und der Bias werden mit Hilfe der Schrittgröße α aktuallisiert.

In [1]:
```
# dw = Weights of the previous timestep
# db = biases of the previous timestep
# m = mean
# v = uncentered variance
# m_dw = mean of the Weights of the previous timestep
# m_db = mean of the biases of the previous timestep
# v_dw = uncentered variance of the Weights of the previous timestep
# v_db = uncentered variance of the biases of the previous timestep
import numpy as np
class AdamOptim():
    # inititalze the adam parameters
    def __init__(self, alpha=0.01, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.m_dw, self.v_dw = 0, 0
        self.m_db, self.v_db = 0, 0
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.alpha = alpha
    def update(self, t, w, b, dw, db):
        ## dw, db are from current minibatch
        ## momentum beta 1
        # weights
        self.m_dw = self.beta1*self.m_dw + (1-self.beta1)*dw
        # biases
        self.m_db = self.beta1*self.m_db + (1-self.beta1)*db

        ## rms beta 2
        # weights
        # update biased second raw moment estimate
        self.v_dw = self.beta2*self.v_dw + (1-self.beta2)*(dw**2)
        # biases
        self.v_db = self.beta2*self.v_db + (1-self.beta2)*(db)

        ## bias correction
        m_dw_corr = self.m_dw/(1-self.beta1**t)
        m_db_corr = self.m_db/(1-self.beta1**t)
        v_dw_corr = self.v_dw/(1-self.beta2**t)
        v_db_corr = self.v_db/(1-self.beta2**t)

        ## update weights and biases
        w = w - self.alpha*(m_dw_corr/(np.sqrt(abs(v_dw_corr))+self.epsilon))
        b = b - self.alpha*(m_db_corr/(np.sqrt(abs(v_db_corr))+self.epsilon))
        return w, b
```

In [2]:
```
def loss_function(m):
    return m**2-2*m+1
## take derivative
def grad_function(m):
    return 2*m-2
# check if the current weight is the same as the previous weight
def check_convergence(w0, w1):
    return (w0 == w1)

# inititialize Adam object
w_0 = 0
b_0 = 0
adam = AdamOptim()
t = 1
```

```python
converged = False

while not converged:
    dw = grad_function(w_0)
    db = grad_function(b_0)
    w_0_old = w_0
    w_0, b_0 = adam.update(t,w=w_0, b=b_0, dw=dw, db=db)
    if check_convergence(w_0, w_0_old):
        print('converged after '+str(t)+' iterations')
        break
    else:
        print('iteration '+str(t)+': weight='+str(w_0))
        t+=1
```

```
iteration 1: weight=0.009999999950000001
iteration 2: weight=0.01999725400385255
iteration 3: weight=0.029989900621600046
iteration 4: weight=0.039976060276935343
iteration 5: weight=0.049953839711732076
iteration 6: weight=0.05992133621693422
iteration 7: weight=0.06987664190678831
iteration 8: weight=0.07981784795404925
iteration 9: weight=0.08974304875491491
iteration 10: weight=0.0996503459940126
iteration 11: weight=0.10953785258172263
iteration 12: weight=0.11940369643843479
iteration 13: weight=0.12924602410293135
iteration 14: weight=0.13906300414491304
iteration 15: weight=0.14885283036466956
iteration 16: weight=0.15861372476597732
iteration 17: weight=0.1683439402914239
iteration 18: weight=0.17804176331244895
iteration 19: weight=0.1877055158694015
iteration 20: weight=0.19733355765979776
iteration 21: weight=0.2069242877756729
iteration 22: weight=0.21647614619342795
iteration 23: weight=0.22598761502184558
iteration 24: weight=0.23545721951596985
iteration 25: weight=0.24488352886630008
iteration 26: weight=0.25426515677423506
iteration 27: weight=0.26360076182591813
iteration 28: weight=0.2728890476775851
iteration 29: weight=0.2821287630662142
iteration 30: weight=0.2913187016597368
iteration 31: weight=0.3004577017613055
iteration 32: weight=0.30954464588215314
iteration 33: weight=0.3185784601974346
iteration 34: weight=0.32755811389914286
iteration 35: weight=0.3364826184597571
iteration 36: weight=0.3453510268197323
iteration 37: weight=0.3541624325113025
iteration 38: weight=0.36291596873035775
iteration 39: weight=0.3716108073673929
iteration 40: weight=0.38024615800772815
iteration 41: weight=0.3888212669103811
iteration 42: weight=0.3973354159741451
iteration 43: weight=0.405787921698609
iteration 44: weight=0.4141781341470465
iteration 45: weight=0.42250543591732403
iteration 46: weight=0.4307692411262221
iteration 47: weight=0.4389689944118521
iteration 48: weight=0.44710416995817326
iteration 49: weight=0.455174270544982
iteration 50: weight=0.4631788266261575
iteration 51: weight=0.47111739543840325
iteration 52: weight=0.47898956014222716
iteration 53: weight=0.4867949289964492
iteration 54: weight=0.4945331345671171
iteration 55: weight=0.502203832971342
iteration 56: weight=0.5098067031562412
iteration 57: weight=0.5173414462128868
iteration 58: weight=0.5248077847249037
iteration 59: weight=0.5322054621511477
iteration 60: weight=0.5395342422417011
iteration 61: weight=0.5467939084862693
iteration 62: weight=0.5539842635939275
iteration 63: weight=0.5611051290030595
iteration 64: weight=0.568156344420244
```

```
iteration 65: weight=0.5751377673867769
iteration 66: weight=0.5820492728714711
iteration 67: weight=0.5888907528883383
iteration 68: weight=0.5956621161377404
iteration 69: weight=0.6023632876695909
iteration 70: weight=0.6089942085671857
iteration 71: weight=0.615554835650261
iteration 72: weight=0.6220451411958915
iteration 73: weight=0.6284651126758695
iteration 74: weight=0.6348147525092407
iteration 75: weight=0.641094077828706
iteration 76: weight=0.6473031202596423
iteration 77: weight=0.653441925710538
iteration 78: weight=0.6595105541736849
iteration 79: weight=0.6655090795350186
iteration 80: weight=0.6714375893920466
iteration 81: weight=0.6772961848788547
iteration 82: weight=0.6830849804972335
iteration 83: weight=0.6888041039530174
iteration 84: weight=0.6944536959967778
iteration 85: weight=0.7000339102680659
iteration 86: weight=0.7055449131424467
iteration 87: weight=0.7109868835806149
iteration 88: weight=0.7163600129789347
iteration 89: weight=0.7216645050207873
iteration 90: weight=0.7269005755281592
iteration 91: weight=0.7320684523129467
iteration 92: weight=0.7371683750274948
iteration 93: weight=0.7422005950139324
iteration 94: weight=0.7471653751519031
iteration 95: weight=0.7520629897043302
iteration 96: weight=0.7568937241608946
iteration 97: weight=0.7616578750789351
iteration 98: weight=0.7663557499215196
iteration 99: weight=0.7709876668924653
iteration 100: weight=0.7755539547681211
iteration 101: weight=0.7800549527257501
iteration 102: weight=0.7844910101683854
iteration 103: weight=0.7888624865460541
iteration 104: weight=0.7931697511732935
iteration 105: weight=0.7974131830429079
iteration 106: weight=0.8015931706359366
iteration 107: weight=0.8057101117278274
iteration 108: weight=0.8097644131908287
iteration 109: weight=0.8137564907926365
iteration 110: weight=0.8176867689913462
iteration 111: weight=0.8215556807267823
iteration 112: weight=0.8253636672082894
iteration 113: weight=0.8291111776990878
iteration 114: weight=0.8327986692973092
iteration 115: weight=0.8364266067138391
iteration 116: weight=0.8399954620471094
iteration 117: weight=0.8435057145549907
iteration 118: weight=0.846957850423946
iteration 119: weight=0.8503523625356184
iteration 120: weight=0.8536897502310298
iteration 121: weight=0.8569705190725782
iteration 122: weight=0.860195180604026
iteration 123: weight=0.8633642521086788
iteration 124: weight=0.8664782563659569
iteration 125: weight=0.8695377214065675
iteration 126: weight=0.8725431802664898
iteration 127: weight=0.8754951707399848
iteration 128: weight=0.8783942351318468
```

```
iteration 129: weight=0.8812409200091125
iteration 130: weight=0.8840357759524461
iteration 131: weight=0.8867793573074179
iteration 132: weight=0.8894722219358927
iteration 133: weight=0.8921149309677457
iteration 134: weight=0.8947080485531188
iteration 135: weight=0.8972521416154303
iteration 136: weight=0.8997477796053487
iteration 137: weight=0.9021955342559357
iteration 138: weight=0.9045959793391636
iteration 139: weight=0.9069496904240053
iteration 140: weight=0.9092572446362935
iteration 141: weight=0.9115192204205393
iteration 142: weight=0.9137361973038962
iteration 143: weight=0.9159087556624508
iteration 144: weight=0.918037476490016
iteration 145: weight=0.9201229411695963
iteration 146: weight=0.9221657312476885
iteration 147: weight=0.9241664282115778
iteration 148: weight=0.9261256132697787
iteration 149: weight=0.9280438671357686
iteration 150: weight=0.9299217698151505
iteration 151: weight=0.9317599003963796
iteration 152: weight=0.933558836845177
iteration 153: weight=0.9353191558027519
iteration 154: weight=0.9370414323879417
iteration 155: weight=0.938726240003377
iteration 156: weight=0.9403741501457696
iteration 157: weight=0.9419857322204138
iteration 158: weight=0.9435615533599866
iteration 159: weight=0.9451021782477242
iteration 160: weight=0.946608168945046
iteration 161: weight=0.9480800847236891
iteration 162: weight=0.9495184819024132
iteration 163: weight=0.950923913688324
iteration 164: weight=0.9522969300228624
iteration 165: weight=0.9536380774324956
iteration 166: weight=0.9549478988841421
iteration 167: weight=0.9562269336453583
iteration 168: weight=0.9574757171493036
iteration 169: weight=0.9586947808645
iteration 170: weight=0.9598846521693933
iteration 171: weight=0.9610458542317184
iteration 172: weight=0.9621789058926664
iteration 173: weight=0.9632843215558446
iteration 174: weight=0.9643626110810171
iteration 175: weight=0.9654142796826064
iteration 176: weight=0.9664398278329352
iteration 177: weight=0.9674397511701782
iteration 178: weight=0.9684145404109944
iteration 179: weight=0.9693646812678026
iteration 180: weight=0.9702906543706602
iteration 181: weight=0.9711929351937009
iteration 182: weight=0.9720719939860842
iteration 183: weight=0.9729282957074055
iteration 184: weight=0.9737622999675106
iteration 185: weight=0.9745744609706605
iteration 186: weight=0.9753652274639815
iteration 187: weight=0.9761350426901402
iteration 188: weight=0.9768843443441767
iteration 189: weight=0.9776135645344266
iteration 190: weight=0.9783231297474617
iteration 191: weight=0.9790134608169765
iteration 192: weight=0.9796849728965454
```

```
iteration 193: weight=0.9803380754361745
iteration 194: weight=0.9809731721625682
iteration 195: weight=0.9815906610630333
iteration 196: weight=0.9821909343729368
iteration 197: weight=0.9827743785666372
iteration 198: weight=0.9833413743518047
iteration 199: weight=0.9838922966670466
iteration 200: weight=0.984427514682753
iteration 201: weight=0.9849473918050764
iteration 202: weight=0.98545228568296
iteration 203: weight=0.985942548218127
iteration 204: weight=0.9864185255779455
iteration 205: weight=0.9868805582110803
iteration 206: weight=0.9873289808658458
iteration 207: weight=0.9877641226111719
iteration 208: weight=0.9881863068600975
iteration 209: weight=0.988595851395703
iteration 210: weight=0.9889930683993977
iteration 211: weight=0.9893782644814731
iteration 212: weight=0.9897517407138409
iteration 213: weight=0.9901137926648667
iteration 214: weight=0.9904647104362185
iteration 215: weight=0.9908047787016447
iteration 216: weight=0.9911342767475994
iteration 217: weight=0.9914534785156336
iteration 218: weight=0.9917626526464719
iteration 219: weight=0.9920620625256936
iteration 220: weight=0.9923519663309407
iteration 221: weight=0.9926326170805743
iteration 222: weight=0.9929042626837034
iteration 223: weight=0.9931671459915103
iteration 224: weight=0.9934215048497993
iteration 225: weight=0.993667572152694
iteration 226: weight=0.993905575897414
iteration 227: weight=0.9941357392400594
iteration 228: weight=0.9943582805523336
iteration 229: weight=0.9945734134791384
iteration 230: weight=0.9947813469969744
iteration 231: weight=0.9949822854730818
iteration 232: weight=0.9951764287252586
iteration 233: weight=0.995363972082295
iteration 234: weight=0.9955451064449625
iteration 235: weight=0.9957200183474998
iteration 236: weight=0.9958888900195372
iteration 237: weight=0.9960518994484044
iteration 238: weight=0.9962092204417675
iteration 239: weight=0.9963610226905409
iteration 240: weight=0.9965074718320245
iteration 241: weight=0.9966487295132145
iteration 242: weight=0.9967849534542412
iteration 243: weight=0.9969162975118857
iteration 244: weight=0.99704291174313
iteration 245: weight=0.9971649424686971
iteration 246: weight=0.9972825323365392
iteration 247: weight=0.9973958203852304
iteration 248: weight=0.9975049421072285
iteration 249: weight=0.9976100295119631
iteration 250: weight=0.9977112111887169
iteration 251: weight=0.997808612369263
iteration 252: weight=0.9979023549902253
iteration 253: weight=0.997992557755128
iteration 254: weight=0.9980793361961048
iteration 255: weight=0.9981628027352366
iteration 256: weight=0.9982430667454899
```

```
iteration 257: weight=0.9983202346112277
iteration 258: weight=0.9983944097882684
iteration 259: weight=0.9984656928634669
iteration 260: weight=0.998534181613794
iteration 261: weight=0.9985999710648938
iteration 262: weight=0.9986631535490955
iteration 263: weight=0.9987238187628614
iteration 264: weight=0.9987820538236523
iteration 265: weight=0.9988379433261909
iteration 266: weight=0.9988915693981099
iteration 267: weight=0.9989430117549652
iteration 268: weight=0.9989923477546039
iteration 269: weight=0.9990396524508698
iteration 270: weight=0.9990849986466379
iteration 271: weight=0.9991284569461627
iteration 272: weight=0.9991700958067327
iteration 273: weight=0.9992099815896199
iteration 274: weight=0.9992481786103167
iteration 275: weight=0.9992847491880505
iteration 276: weight=0.9993197536945726
iteration 277: weight=0.9993532506022106
iteration 278: weight=0.9993852965311832
iteration 279: weight=0.999415946296171
iteration 280: weight=0.9994452529521384
iteration 281: weight=0.9994732678394059
iteration 282: weight=0.999500040627969
iteration 283: weight=0.9995256193610621
iteration 284: weight=0.9995500504979664
iteration 285: weight=0.9995733789560621
iteration 286: weight=0.9995956481521241
iteration 287: weight=0.9996169000428623
iteration 288: weight=0.9996371751647086
iteration 289: weight=0.9996565126728514
iteration 290: weight=0.9996749503795203
iteration 291: weight=0.9996925247915246
iteration 292: weight=0.9997092711470486
iteration 293: weight=0.9997252234517061
iteration 294: weight=0.9997404145138615
iteration 295: weight=0.9997548759792192
iteration 296: weight=0.9997686383646873
iteration 297: weight=0.9997817310915225
iteration 298: weight=0.9997941825177595
iteration 299: weight=0.9998060199699335
iteration 300: weight=0.9998172697740993
iteration 301: weight=0.9998279572861581
iteration 302: weight=0.9998381069214939
iteration 303: weight=0.9998477421839309
iteration 304: weight=0.9998568856940179
iteration 305: weight=0.9998655592166464
iteration 306: weight=0.9998737836880133
iteration 307: weight=0.9998815792419323
iteration 308: weight=0.9998889652355071
iteration 309: weight=0.9998959602741706
iteration 310: weight=0.9999025822361022
iteration 311: weight=0.9999088482960293
iteration 312: weight=0.9999147749484246
iteration 313: weight=0.999920378030106
iteration 314: weight=0.9999256727422497
iteration 315: weight=0.9999306736718255
iteration 316: weight=0.9999353948124632
iteration 317: weight=0.9999398495847598
iteration 318: weight=0.9999440508560381
iteration 319: weight=0.9999480109595638
iteration 320: weight=0.9999517417132329
```

```
iteration 321: weight=0.9999552544377384
iteration 322: weight=0.9999585599742249
iteration 323: weight=0.9999616687014424
iteration 324: weight=0.9999645905524075
iteration 325: weight=0.999967335030582
iteration 326: weight=0.9999699112255794
iteration 327: weight=0.9999723278284075
iteration 328: weight=0.9999745931462574
iteration 329: weight=0.9999767151168482
iteration 330: weight=0.9999787013223368
iteration 331: weight=0.9999805590028027
iteration 332: weight=0.9999822950693158
iteration 333: weight=0.9999839161165983
iteration 334: weight=0.9999854284352881
iteration 335: weight=0.9999868380238144
iteration 336: weight=0.9999881505998927
iteration 337: weight=0.9999893716116505
iteration 338: weight=0.9999905062483903
iteration 339: weight=0.9999915594509997
iteration 340: weight=0.9999925359220176
iteration 341: weight=0.9999934401353642
iteration 342: weight=0.9999942763457434
iteration 343: weight=0.9999950485977274
iteration 344: weight=0.9999957607345287
iteration 345: weight=0.999996416406471
iteration 346: weight=0.9999970190791647
iteration 347: weight=0.9999975720413955
iteration 348: weight=0.9999980784127344
iteration 349: weight=0.9999985411508757
iteration 350: weight=0.9999989630587119
iteration 351: weight=0.9999993467911512
iteration 352: weight=0.9999996948616862
iteration 353: weight=1.0000000096487203
iteration 354: weight=1.0000002934016596
iteration 355: weight=1.0000005482467753
iteration 356: weight=1.0000007761928456
iteration 357: weight=1.0000009791365825
iteration 358: weight=1.0000011588678495
iteration 359: weight=1.0000013170746784
iteration 360: weight=1.0000014553480892
iteration 361: weight=1.0000015751867204
iteration 362: weight=1.0000016780012766
iteration 363: weight=1.0000017651187962
iteration 364: weight=1.0000018377867486
iteration 365: weight=1.0000018971769635
iteration 366: weight=1.0000019443894
iteration 367: weight=1.0000019804557585
iteration 368: weight=1.0000020063429436
iteration 369: weight=1.0000020229563793
iteration 370: weight=1.0000020311431854
iteration 371: weight=1.0000020316952167
iteration 372: weight=1.0000020253519715
iteration 373: weight=1.0000020128033733
iteration 374: weight=1.0000019946924315
iteration 375: weight=1.0000019716177821
iteration 376: weight=1.0000019441361176
iteration 377: weight=1.0000019127645048
iteration 378: weight=1.000001877982599
iteration 379: weight=1.0000018402347561
iteration 380: weight=1.0000017999320474
iteration 381: weight=1.000001757454179
iteration 382: weight=1.0000017131513226
iteration 383: weight=1.0000016673458578
iteration 384: weight=1.000001620334032
```

```
iteration 385: weight=1.0000015723875386
iteration 386: weight=1.0000015237550193
iteration 387: weight=1.0000014746634915
iteration 388: weight=1.0000014253197047
iteration 389: weight=1.0000013759114286
iteration 390: weight=1.0000013266086762
iteration 391: weight=1.0000012775648637
iteration 392: weight=1.0000012289179103
iteration 393: weight=1.0000011807912805
iteration 394: weight=1.0000011332949712
iteration 395: weight=1.000001086526446
iteration 396: weight=1.000001040571519
iteration 397: weight=1.0000009955051905
iteration 398: weight=1.0000009513924366
iteration 399: weight=1.0000009082889536
iteration 400: weight=1.0000008662418622
iteration 401: weight=1.0000008252903696
iteration 402: weight=1.0000007854663946
iteration 403: weight=1.0000007467951555
iteration 404: weight=1.000000709295723
iteration 405: weight=1.0000006729815407
iteration 406: weight=1.0000006378609128
iteration 407: weight=1.0000006039374625
iteration 408: weight=1.0000005712105615
iteration 409: weight=1.0000005396757328
iteration 410: weight=1.0000005093250262
iteration 411: weight=1.0000004801473712
iteration 412: weight=1.0000004521289048
iteration 413: weight=1.0000004252532784
iteration 414: weight=1.0000003995019433
iteration 415: weight=1.000000374854416
iteration 416: weight=1.0000003512885252
iteration 417: weight=1.000000328780641
iteration 418: weight=1.0000003073058867
iteration 419: weight=1.0000002868383355
iteration 420: weight=1.0000002673511916
iteration 421: weight=1.000000248816957
iteration 422: weight=1.000000231207586
iteration 423: weight=1.0000002144946256
iteration 424: weight=1.0000001986493454
iteration 425: weight=1.0000001836428556
iteration 426: weight=1.0000001694462146
iteration 427: weight=1.0000001560305274
iteration 428: weight=1.0000001433670334
iteration 429: weight=1.0000001314271876
iteration 430: weight=1.0000001201827318
iteration 431: weight=1.0000001096057591
iteration 432: weight=1.000000099668772
iteration 433: weight=1.000000090344732
iteration 434: weight=1.000000081607105
iteration 435: weight=1.0000000734299008
iteration 436: weight=1.0000000657877053
iteration 437: weight=1.0000000586557103
iteration 438: weight=1.0000000520097374
iteration 439: weight=1.0000000458262583
iteration 440: weight=1.0000000400824103
iteration 441: weight=1.000000034756009
iteration 442: weight=1.0000000298255576
iteration 443: weight=1.0000000252702532
iteration 444: weight=1.0000000210699902
iteration 445: weight=1.0000000172053607
iteration 446: weight=1.0000000136576537
iteration 447: weight=1.0000000104088511
iteration 448: weight=1.000000007441623
```

```
iteration 449: weight=1.0000000047393205
iteration 450: weight=1.0000000022859659
iteration 451: weight=1.0000000000662446
iteration 452: weight=0.9999999980654928
iteration 453: weight=0.9999999962696857
iteration 454: weight=0.9999999946654243
iteration 455: weight=0.9999999932399216
iteration 456: weight=0.9999999919809877
iteration 457: weight=0.9999999908770149
iteration 458: weight=0.999999989916962
iteration 459: weight=0.9999999890903385
iteration 460: weight=0.9999999883871876
iteration 461: weight=0.9999999877980703
iteration 462: weight=0.9999999873140484
iteration 463: weight=0.9999999869266677
iteration 464: weight=0.9999999866279413
iteration 465: weight=0.9999999864103329
iteration 466: weight=0.9999999862667397
iteration 467: weight=0.9999999861904763
iteration 468: weight=0.9999999861752578
iteration 469: weight=0.9999999862151839
iteration 470: weight=0.9999999863047233
iteration 471: weight=0.9999999864386969
iteration 472: weight=0.9999999866122634
iteration 473: weight=0.9999999868209039
iteration 474: weight=0.9999999870604066
iteration 475: weight=0.9999999873268534
iteration 476: weight=0.9999999876166047
iteration 477: weight=0.9999999879262865
iteration 478: weight=0.9999999882527769
iteration 479: weight=0.999999988593193
iteration 480: weight=0.9999999889448784
iteration 481: weight=0.9999999893053915
iteration 482: weight=0.9999999896724932
iteration 483: weight=0.9999999900441359
iteration 484: weight=0.9999999904184526
iteration 485: weight=0.9999999907937464
iteration 486: weight=0.9999999911684802
iteration 487: weight=0.9999999915412672
iteration 488: weight=0.9999999919108616
iteration 489: weight=0.9999999922761498
iteration 490: weight=0.9999999926361411
iteration 491: weight=0.9999999929899606
iteration 492: weight=0.9999999933368405
iteration 493: weight=0.9999999936761135
iteration 494: weight=0.9999999940072047
iteration 495: weight=0.9999999943296259
iteration 496: weight=0.9999999946429682
iteration 497: weight=0.9999999949468965
iteration 498: weight=0.9999999952411437
iteration 499: weight=0.999999995525505
iteration 500: weight=0.9999999957998327
iteration 501: weight=0.9999999960640314
iteration 502: weight=0.9999999963180537
iteration 503: weight=0.999999996561895
iteration 504: weight=0.9999999967955902
iteration 505: weight=0.9999999970192094
iteration 506: weight=0.9999999972328546
iteration 507: weight=0.9999999974366559
iteration 508: weight=0.9999999976307687
iteration 509: weight=0.9999999978153705
iteration 510: weight=0.9999999979906583
iteration 511: weight=0.9999999981568457
iteration 512: weight=0.999999998314161
```

```
iteration 513: weight=0.9999999984628445
iteration 514: weight=0.9999999986031467
iteration 515: weight=0.9999999987353264
iteration 516: weight=0.9999999988596489
iteration 517: weight=0.9999999989763841
iteration 518: weight=0.9999999990858058
iteration 519: weight=0.9999999991881894
iteration 520: weight=0.9999999992838112
iteration 521: weight=0.9999999993729474
iteration 522: weight=0.9999999994558726
iteration 523: weight=0.9999999995328595
iteration 524: weight=0.9999999996041773
iteration 525: weight=0.9999999996700918
iteration 526: weight=0.9999999997308642
iteration 527: weight=0.9999999997867507
iteration 528: weight=0.999999999838002
iteration 529: weight=0.9999999998848629
iteration 530: weight=0.9999999999275718
iteration 531: weight=0.9999999999663607
iteration 532: weight=1.0000000000014544
iteration 533: weight=1.0000000000330709
iteration 534: weight=1.0000000000614209
iteration 535: weight=1.0000000000867075
iteration 536: weight=1.0000000001091267
iteration 537: weight=1.000000000128867
iteration 538: weight=1.0000000001461087
iteration 539: weight=1.0000000001610256
iteration 540: weight=1.0000000001737834
iteration 541: weight=1.0000000001845408
iteration 542: weight=1.0000000001934488
iteration 543: weight=1.0000000002006517
iteration 544: weight=1.0000000002062863
iteration 545: weight=1.0000000002104827
iteration 546: weight=1.0000000002133642
iteration 547: weight=1.0000000002150473
iteration 548: weight=1.0000000002156426
iteration 549: weight=1.0000000002152538
iteration 550: weight=1.0000000002139788
iteration 551: weight=1.0000000002119098
iteration 552: weight=1.0000000002091332
iteration 553: weight=1.0000000002057299
iteration 554: weight=1.0000000002017755
iteration 555: weight=1.0000000001973406
iteration 556: weight=1.000000000192491
iteration 557: weight=1.0000000001872875
iteration 558: weight=1.0000000001817868
iteration 559: weight=1.0000000001760414
iteration 560: weight=1.0000000001700993
iteration 561: weight=1.000000000164005
iteration 562: weight=1.0000000001577996
iteration 563: weight=1.00000000015152
iteration 564: weight=1.0000000001452
iteration 565: weight=1.000000000138871
iteration 566: weight=1.0000000001325604
iteration 567: weight=1.0000000001262936
iteration 568: weight=1.000000000120093
iteration 569: weight=1.0000000001139786
iteration 570: weight=1.000000000107968
iteration 571: weight=1.0000000001020772
iteration 572: weight=1.0000000000963196
iteration 573: weight=1.0000000000907068
iteration 574: weight=1.000000000085249
iteration 575: weight=1.0000000000799543
iteration 576: weight=1.0000000000748297
```

```
iteration 577: weight=1.0000000000698805
iteration 578: weight=1.0000000000651112
iteration 579: weight=1.0000000000605247
iteration 580: weight=1.0000000000561229
iteration 581: weight=1.000000000051907
iteration 582: weight=1.000000000047877
iteration 583: weight=1.0000000000440323
iteration 584: weight=1.0000000000403715
iteration 585: weight=1.0000000000368925
iteration 586: weight=1.0000000000335927
iteration 587: weight=1.000000000030469
iteration 588: weight=1.0000000000275175
iteration 589: weight=1.0000000000247347
iteration 590: weight=1.000000000022116
iteration 591: weight=1.0000000000196572
iteration 592: weight=1.000000000017353
iteration 593: weight=1.0000000000151987
iteration 594: weight=1.000000000013189
iteration 595: weight=1.0000000000113185
iteration 596: weight=1.000000000009582
iteration 597: weight=1.0000000000079738
iteration 598: weight=1.0000000000064888
iteration 599: weight=1.0000000000051212
iteration 600: weight=1.0000000000038658
iteration 601: weight=1.0000000000027172
iteration 602: weight=1.0000000000016698
iteration 603: weight=1.0000000000007185
iteration 604: weight=0.9999999999998581
iteration 605: weight=0.9999999999990835
iteration 606: weight=0.9999999999983898
iteration 607: weight=0.9999999999977722
iteration 608: weight=0.999999999997226
iteration 609: weight=0.9999999999967466
iteration 610: weight=0.9999999999963296
iteration 611: weight=0.9999999999959708
iteration 612: weight=0.999999999995666
iteration 613: weight=0.9999999999954114
iteration 614: weight=0.9999999999952033
iteration 615: weight=0.999999999995038
iteration 616: weight=0.999999999994912
iteration 617: weight=0.999999999994822
iteration 618: weight=0.999999999994765
iteration 619: weight=0.9999999999947379
iteration 620: weight=0.999999999994738
iteration 621: weight=0.9999999999947625
iteration 622: weight=0.999999999994809
iteration 623: weight=0.9999999999948752
iteration 624: weight=0.9999999999949587
iteration 625: weight=0.9999999999950575
iteration 626: weight=0.9999999999951696
iteration 627: weight=0.9999999999952933
iteration 628: weight=0.9999999999954268
iteration 629: weight=0.9999999999955685
iteration 630: weight=0.9999999999957171
iteration 631: weight=0.9999999999958711
iteration 632: weight=0.9999999999960292
iteration 633: weight=0.9999999999961904
iteration 634: weight=0.9999999999963536
iteration 635: weight=0.9999999999965179
iteration 636: weight=0.9999999999966823
iteration 637: weight=0.9999999999968462
iteration 638: weight=0.9999999999970088
iteration 639: weight=0.9999999999971695
iteration 640: weight=0.9999999999973277
```

```
iteration 641: weight=0.9999999999974829
iteration 642: weight=0.9999999999976348
iteration 643: weight=0.9999999999977829
iteration 644: weight=0.9999999999979269
iteration 645: weight=0.9999999999980665
iteration 646: weight=0.9999999999982015
iteration 647: weight=0.9999999999983318
iteration 648: weight=0.9999999999984571
iteration 649: weight=0.9999999999985775
iteration 650: weight=0.9999999999986927
iteration 651: weight=0.9999999999988028
iteration 652: weight=0.9999999999989078
iteration 653: weight=0.9999999999990076
iteration 654: weight=0.9999999999991023
iteration 655: weight=0.999999999999192
iteration 656: weight=0.9999999999992767
iteration 657: weight=0.9999999999993565
iteration 658: weight=0.9999999999994316
iteration 659: weight=0.9999999999995018
iteration 660: weight=0.9999999999995676
iteration 661: weight=0.999999999999629
iteration 662: weight=0.999999999999686
iteration 663: weight=0.999999999999739
iteration 664: weight=0.999999999999788
iteration 665: weight=0.9999999999998331
iteration 666: weight=0.9999999999998747
iteration 667: weight=0.9999999999999126
iteration 668: weight=0.9999999999999473
iteration 669: weight=0.9999999999999788
iteration 670: weight=1.0000000000000073
iteration 671: weight=1.0000000000000329
iteration 672: weight=1.0000000000000557
iteration 673: weight=1.0000000000000762
iteration 674: weight=1.0000000000000941
iteration 675: weight=1.00000000000011
iteration 676: weight=1.0000000000001235
iteration 677: weight=1.0000000000001352
iteration 678: weight=1.000000000000145
iteration 679: weight=1.0000000000001532
iteration 680: weight=1.0000000000001599
iteration 681: weight=1.000000000000165
iteration 682: weight=1.0000000000001688
iteration 683: weight=1.0000000000001714
iteration 684: weight=1.000000000000173
iteration 685: weight=1.0000000000001734
iteration 686: weight=1.000000000000173
iteration 687: weight=1.0000000000001716
iteration 688: weight=1.0000000000001696
iteration 689: weight=1.000000000000167
iteration 690: weight=1.0000000000001639
iteration 691: weight=1.00000000000016
iteration 692: weight=1.0000000000001559
iteration 693: weight=1.0000000000001514
iteration 694: weight=1.0000000000001465
iteration 695: weight=1.0000000000001414
iteration 696: weight=1.0000000000001361
iteration 697: weight=1.0000000000001306
iteration 698: weight=1.000000000000125
iteration 699: weight=1.0000000000001195
iteration 700: weight=1.0000000000001137
iteration 701: weight=1.000000000000108
iteration 702: weight=1.0000000000001021
iteration 703: weight=1.0000000000000966
iteration 704: weight=1.000000000000091
```

```
iteration 705: weight=1.0000000000000855
iteration 706: weight=1.0000000000000802
iteration 707: weight=1.0000000000000748
iteration 708: weight=1.0000000000000697
iteration 709: weight=1.0000000000000648
iteration 710: weight=1.00000000000006
iteration 711: weight=1.0000000000000553
iteration 712: weight=1.0000000000000508
iteration 713: weight=1.0000000000000466
iteration 714: weight=1.0000000000000426
iteration 715: weight=1.0000000000000386
iteration 716: weight=1.0000000000000349
iteration 717: weight=1.0000000000000313
iteration 718: weight=1.000000000000028
iteration 719: weight=1.0000000000000249
iteration 720: weight=1.000000000000022
iteration 721: weight=1.000000000000019
iteration 722: weight=1.0000000000000164
iteration 723: weight=1.000000000000014
iteration 724: weight=1.0000000000000118
iteration 725: weight=1.0000000000000098
iteration 726: weight=1.0000000000000078
iteration 727: weight=1.000000000000006
iteration 728: weight=1.0000000000000044
iteration 729: weight=1.0000000000000029
iteration 730: weight=1.0000000000000016
iteration 731: weight=1.0000000000000002
iteration 732: weight=0.9999999999999991
iteration 733: weight=0.9999999999999981
iteration 734: weight=0.9999999999999972
iteration 735: weight=0.9999999999999964
iteration 736: weight=0.9999999999999958
iteration 737: weight=0.9999999999999952
iteration 738: weight=0.9999999999999947
iteration 739: weight=0.9999999999999942
iteration 740: weight=0.9999999999999939
iteration 741: weight=0.9999999999999936
iteration 742: weight=0.9999999999999933
iteration 743: weight=0.9999999999999931
iteration 744: weight=0.999999999999993
iteration 745: weight=0.9999999999999929
converged after 746 iterations
```

Einsatzbereich

Der Adam Optimierer wurde entwickelt, um den Lernprozess neuronaler Netzwerke zu optimieren Bushaev, 2018. Adam kann zum Beispiel dafür verwendet werden, um Pixeländerungen an Bildern durchzuführen, ohne den Hashwert zu verändern. Struppek et al. Struppek et al., 2022, S. 6f konten zeigen, das mit Adam Hash Kollisionen forciert oder vermieden werden können. Dadurch ist es unter anderem möglich kleine Änderungen an den Pixeln von Bildern vorzunehmen, ohne den Hash-Wert zu ändern. Apple verwendet den Hash-Wert von Bildern um Kinderpornografie in Bildern von Nutzern der iCloud zu identifizieren. Mit Hilfe von Optimierern, wie Adam, können Bildinformationen manipuliert und Personen belastende Bilder untergeschoben werden, deren Hash-Wert mit einem Hash-Wert aus einer Child Sexual Abuse Material (CSAM) Datenbank übereinstimmt Struppek et al., 2022, S. 1.

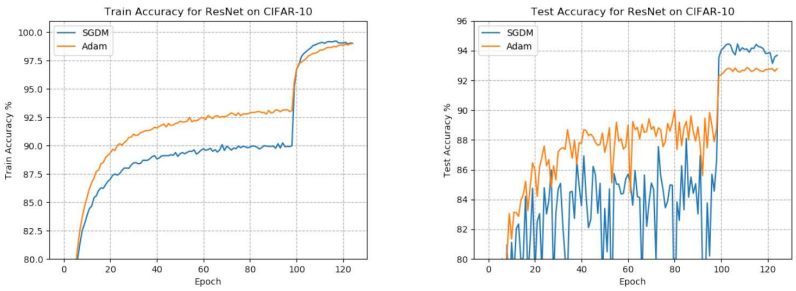
Jais et al. Jais et al., 2019, S. 1 konnten zeigen, das Adam in tiefen und weiten neuronalen Netzwerken verwendet werden kann, um deren Leistung zu verbessern. Als Grundlage für das tiefe und weite neuronale Netzwerk wurden Daten zu Brustkrebs verwendet. Diese wurden korelliert, um Muster im Datensatz zu

erkennen. Daraufhin wurde mit den Daten das tiefe und weite neuronale Netzwerk trainiert. Der Trainingsprozess wurde mit Adam optimiert Jais et al., 2019, S. 43f..

Limitierungen

Adam ist einer der Standard-Algorithmen, wenn es darum geht, adaptive Gradientenmethoden in Deep-Learning einzusetzen Tang et al., 2019, S. 1f. Jedoch zeigt sich, dass der Adam Optimierer und weitere adaptive Optimierungsmethoden im Vergleich mit dem stochastischen Gradientenabstieg (SGD) nicht so gut generalisieren. Als Generalisierung wird die Übertragung der aus den Trainingsdaten gefundenen Erkenntnisse auf Realdaten bezeichnet. Adam schneidet bei Trainingsdaten gut ab. Bei Testdaten liefert jedoch SGD bessere Ergebnisse. Der Leistungsunterschied zwischen SGD und Adam, die Generalisierungslücke genannt, liegt bei zwei ProzentTang et al., 2019, S. 1f.

In [5]:
```
from IPython.display import display, Image
display(Image(filename="Tang-et-al.png", height=400, width=400))
```



(a) train (b) test

Abbildung: Training der ResNet-34-Architektur auf dem CIFAR-10-Datensatz mit stochastischem Gradientenabstieg (SGD) und Adam. Adam hat eine schnellere anfängliche Konvergenzgeschwindigkeit, aber die endgültige Testgenauigkeit ist niedriger als die von SGD Tang et al., 2019, S. 1f.

Zum besseren Verständnis werden im Folgenden wichtige Aspekte von SGD und ADAM beschrieben. SGD berechnet nur auf einer kleinen Teilmenge einer zufälligen Auswahl von Datenbeispielen den Gradientenabstieg. Das hat zur Folge, dass die Lernrate niedriger ist, die "Leistung" entspricht allerdings der eines normalen Gradientenabstiegs. Im direkten Vergleich dazu berechnet ADAM die Lernrate für verschiedene Parameter aus Schätzungen des ersten und zweiten Momentums. Dabei werden die Vorteile von RMSprop und AdaGrad kombiniert, um adaptive Lernraten für verschiedene Parameter zu berechnen.

Kaur et al. (Kaur, 2022, Kaur, 2022, Poster, 2020-NIPS-SGD-poster) haben folgende Problembereiche bei Adam identifiziert, die für die schlechte Generalisierungsfähigkeit verantwortlich sind:

- **Ungleiche Skalierung der Gradienten**
  Das kann zur Folge haben, dass für adaptive Gradientenmethoden eine schlechte Generalisierungsleistung zustande kommt. SGD skaliert glechmäßiger und etwaige Trainingsfehler können besser verallgemeinert werden.
- **Exponentiell gleitender Durchschnitt**
  In Adam wird ein sogenannter exponentiell gleitender Durchschnitt eingesetzt, der die Lernrate nicht monoton sinken lassen kann. Die Folge daraus ist eine suboptimale Lösung und eine schlechte Generalisierungsleistung.
- **Unter Umständen zu geringe Lernrate**
  Wenn Adam mit einer zu niedrigen Lernrate konfiguriert wird, kommt es zu keiner effektiven

Konvergenz. Dadurch findet Adam nicht den richtigen Pfand, wodurch der Optimierer zu einem supotimalen Punkt konvergiert.

- **Unter Umständen stark steigende Lernrate**
  Um zu konvergieren erhöht Adam manchmal die Lernrate. Tritt dieser Fall ein, so wirkt sich das negativ auf die Leistung des Algorithmus aus.

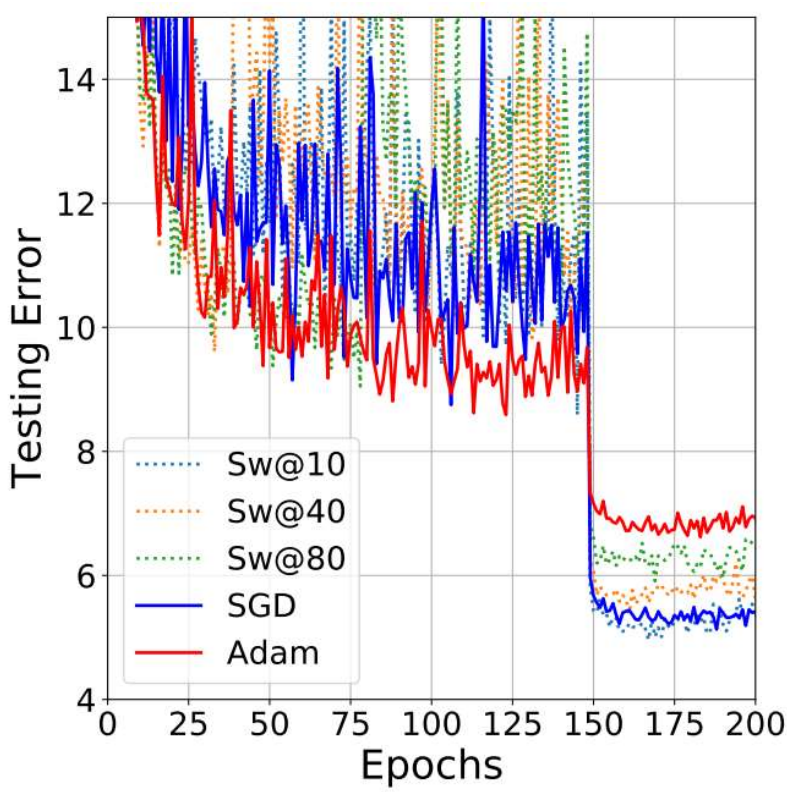
Allerdings können die Grenzen von ADAM durch Einsatz der Strategie SWATS überwunden werden. Nitish Shirish Keskar und Richard Socher haben untersucht, wie die Generalisierungsleistung durch den Wechsel von ADAM zu SGD verbessert werden kann Bushaev, 2018, Problems with Adam. Im dazugehörigen Paper wird die SWATS Strategie vorgestellt, was so viel bedeutet wie "**Sw** from **A**dam **t**o **S**GD". Das heißt, dass nur zu SGD gewechselt wird, wenn eine Bedingung eintritt Keskar et al., 2020, S. 2ff. Die Bedingung ist folgendermaßen aufgebaut:

$$\left| \frac{\lambda_k}{1-\beta_2^k} - \gamma_k \right| < \epsilon,$$

Dabei wird der verzerrungskorrigierte exponentieller Mittelwert mit dem aktuellen Wert γ_k verglichen. Die Bias-Korrektur ist notwendig, um während dem Training zu verhindern, dass die Null-Initialisierung einen Einfluss hat. Wenn diese Bedingung erfüllt ist, wird die Lernrate von SGD mit $\Lambda := \frac{\lambda_k}{(1-\beta_2^k)}$ gewählt Keskar et al., 2020, S. 4f.

In nachfolgendem Diagramm ist zu sehen, wie sich der Switch zu SGD je nach fortgeschrittener Epoche auswirkt.

In [6]:
```
from IPython.display import display, Image
display(Image(filename="Epochs.png", height=400, width=400))
```



Dabei wird vermehrt darauf hingewiesen, dass der Switch zum richtigen Zeitpunkt erfolgen muss. Wenn zu spät zu SGD gewechselt wird, erhält man ähnlich wie bei ADAM eine Generalisierungslücke. Ein früher Wechsel führt zu einer mit SGD vergleichbaren Testgenauigkeit Keskar et al., 2020, S. 3f.

Alternativen & Abgrenzung

Im folgenden Abschnitt wird Adam von anderen Optimierungsmethoden abgegrenzt. Es werden folgende Optimierungsmethoden besprochen:

- RMSProp - Tielemann & Hinton, 2012
- AdaGrad - Duchi et al., 2011
- vSGD - Schaul et al., 2012
- AdaDelta - Zeiler, 2012
- natürliche Newton-Methode - Roux & Fitzgibbon, 2010
- SFO - Sohl-Dicksein et al., 2014
- NGD - Amari, 1998

vSGD, AdaDelta und die natürliche Newton-Methode schätzen die Krümmung aus Informationen erster Ordnung und setzen anhand dieser Informationen die Schrittweite. Der SFO (Sum-of-Functions Optimierer) ist ein quasi-Newton-Verfahren, das auch Minibatches aufbaut. Im Gegensatz zum Adam Optimierer hat der SFO einen linearen Speicherbedarf in der Anzahl der Minibatch-Partitionen eines Datensatzen. SFO ist deswegen auf speicherbeschränkten Systemen, wie zum Beispiel einer GPU, meist nicht ausführbar.

Bei RMSProp (Root Mean Square Propagation) handelt es sich um einen Optimierer, der eng mit dem Adam Optimierer verwandt ist. Es kann auch vorkommen, dass eine Version mit Momentum eingesetzt wird Graves, 2013. Wird RMSProp mit Momentum eingesetzt, so werden die Parameteraktualisierungen anhand eines Momentums auf dem neu skalierten Gradienten generiert. Beim Adam Optimierer werden die Aktualisierungen anhand eines Durchschnitts des ersten und zweiten Momentums des Gradienten geschätzt. Bei RMSProb wird der Bias nicht korrigiert. Eine Biaskorrektur wäre jedoch notwendig, wenn ein Wert von β_2 nahe dem Wert eins (1) ist. In diesem Fall kann eine Nichtkorrektur des Bias zu sehr großen Schrittweiten führen, woraus sich eine Divergenz entwickeln kann Diederik et al., 2017, S. 5,8 [siehe 6.4]. RMSProp und Adadelta zeilen sich den gleichen Update-Vektor:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

Der AdaGrad Algorithmus passt die Lernrate an die Parameter an und führt kleine Aktualisierungen für Parameter durch, die mit häufig auftretenden Features verbunden sind. Größere Aktualisierungen werden für Parameter durchgeführt, die mit seltenen Features verbunden sind Ruder, 2016; Dean et al., 2012.

Fazit

Der Adam Optimierer kombiniert die positiven Eigenschaften von AdaGrad und RMSProb. Also die effiziente Verarbeitung von "Sparse Gradients" von AdaGrad und die gute Performance in nicht stationären Umgebungen von RMSProb. Adam hat unter Verwendung der Standardparameter eine gute Performance und Geschwindigkeit und kann mit "Sparse Gradients" umgehen. Leider ist die Generalisierungsfähigkeit von Adam geringer als das Standardverfahren SGD. Das liegt primär an der, im Vergleich zu SGD, ungleichen Skalierung der Gradienten. Außerdem leidet die Leistung von Adam sehr stark an zu gering gewählten Lernraten, wodurch Adam zu supoptimalen Punkten konvergiert. Diese Nachteile können durch eine

Kombination aus Adam und SGD umgangen werden. Die SWATS Strategie verwendet Adam am Anfang für Optimierungen und wechselt später zu SGD für eine bessere Leistung des trainierten Modells auf Realdaten.

Der Adam Optimierer ist ein guter Standard Optimierer, welcher in vielen Situationen angewandt werden kann. Jedoch sollten Nutzern die Limitierungen von Adam bekannt sein, um bestmögliche Ergebnisse zu erzielen. Im Zweifelsfall bietet sich eine Kombination mit SGD (SWATS Strategie) oder ein alleiniges Verwenden von SGD für die beste Leistung des trainierten Modells an.

Quellen

Eingesetzte Quellen

1.

Title:	Mathematische Statistik
Authors:	Ludger Rüschendorf
Date:	2014
Source:	https://link.springer.com/book/10.1007/978-3-642-41997-3
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Introduction to Loss Functions
Authors:	DataRobot
Date:	30. April 2018
Source:	https://www.datarobot.com/blog/introduction-to-loss-functions/
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Was ist ein Machine Learning-Modell?
Authors:	QuinnRadich, v-alje, eliotcowley
Date:	2. Dezember 2022
Source:	https://learn.microsoft.com/de-de/windows/ai/windows-ml/what-is-a-machine-learning-model
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Interpretable machine learning: definitions, methods, and applications
Authors:	James W. Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, Bin Yu
Date:	14. Januar 2019
Source:	https://arxiv.org/pdf/1901.04592.pdf?fbclid=IwAR2frcHrhLc4iaH5-TmKKq263NVvAKHtG4uQoiVNDeLAG3QFzdje-yzZjiQ
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Gradienten berechnen
Authors:	studyflix
Date:	-- keine Angabe --
Source:	https://studyflix.de/mathematik/gradient-berechnen-1350
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Introduction to Optimizers
Authors:	DataRobot
Date:	7. Mai 2018
Source:	https://www.datarobot.com/blog/introduction-to-optimizers/
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Step 1: Understand how Adam works
Authors:	Enoch Kan
Date:	6. November 2020
Source:	https://towardsdatascience.com/how-to-implement-an-adam-optimizer-from-scratch-76e7b217f1cc https://miro.medium.com/max/390/0*F9hIO3J8_i2F0p3y.webp
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Adam: A Method for Stochastic Optimization
Authors:	Diederik P. Kingma, Jimmy Ba
Date:	30. Januar 2017
Source:	https://arxiv.org/abs/1412.6980
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Adaptive Subgradient Methods for Online Learning and Stochastic Optimization
Authors:	John Duchi, Elad Hazan, Yoram Singer
Date:	11 Juli 2011
Source:	https://dl.acm.org/doi/pdf/10.5555/1953048.2021068
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Momentum Optimizer in Deep Learning | Explained in Detail
Authors:	Coding Lane

Date:	21. August 2021
Source:	https://youtu.be/Vce8w1sy0e8
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	RMSprop Optimizer Explained in Detail | Deep Learning
Authors:	Coding Lane
Date:	27. August 2021
Source:	https://youtu.be/ajl_HTyaCu8
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Titel:	An overview of gradient descent optimization algorithms
Autor:	Sebastian Ruder
Datum:	19 January 2016
Source:	https://ruder.io/optimizing-gradient-descent/index.html#adagrad
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Generating Sequences With Recurrent Neural Networks
Authors:	Alex Graves
Date:	5. Juni 2014
Source:	https://arxiv.org/abs/1308.0850
Zuletzt aufgerufen am:	13. Dezember 2022

1.

Title:	Adam — latest trends in deep learning optimization.
Authors:	Vitaly Bushaev
Date:	22. October 2018
Source:	https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
Zuletzt aufgerufen am:	14. Dezember 2022

1.

Title:	Improving Generalization Performance by Switching from Adam to SGD
Authors:	Nitish Shirish Keskar, Richard Socher
Date:	20. Dezember 2020
Source:	https://arxiv.org/abs/1712.07628
Zuletzt aufgerufen am:	14. Dezember 2022

1. Title: Learning to Break Deep Perceptual Hashing: The Use Case NeuralHash

Authors:	Lukas Struppek, Dominik Hintersdorf, Daniel Neider, Kristian Kersting
Date:	9. Juni 2022
Source:	https://arxiv.org/pdf/2111.06628.pdf
Zuletzt aufgerufen am:	14. Dezember 2022

1.

Title:	Adam Optimization Algorithm for Wide and Deep Neural Network
Authors:	Imran Khan Mohd Jais, Amelia Ritahani Ismail, Syed Qamrun Nisa
Date:	23. Juni 2019
Source:	http://journal2.um.ac.id/index.php/keds/article/view/6775/0
Zuletzt aufgerufen am:	14. Dezember 2022

1.

Title:	Overview of mini-batch gradient descent
Authors:	Geoffrey Hinton, Nitish Srivastava, Kevin Swersky
Date:	2012
Source:	https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
Zuletzt aufgerufen am:	14. Dezember 2022

1.

Title:	A Bounded Scheduling Method for Adaptive Gradient Methods
Authors:	Mingxing Tang, Zhen Huang, Yuan Yuan, Changjian Wang, Yuxing Peng
Date:	1. September 2019
Source:	https://www.mdpi.com/2076-3417/9/17/3569
Zuletzt aufgerufen am:	16. Dezember 2022

1.

Title:	Why Should adam Optimizer Not be the Default Learning Algorithm?
Authors:	Harjot Kaur
Date:	20. August 2022
Source:	https://pub.towardsai.net/why-adam-optimizer-should-not-be-the-default-learning-algorithm-a2b8d019eaa0
Zuletzt aufgerufen am:	16. Dezember 2022

1.

Title:	Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning
Authors:	Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven HOI, Weinan E
Date:	29. November 2021
Source:	https://panzhous.github.io/assets/pdf/2020-NIPS-SGD-poster.pdf; (https://panzhous.github.io/#2020)
Zuletzt aufgerufen am:	16. Dezember 2022

1.

Title:	Large Scale Distributed Deep Networks
Authors:	Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng
Date:	2012
Source:	https://proceedings.neurips.cc/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf; (https://proceedings.neurips.cc/papers/search?q=large+scale+distributed+deep+networks)
Zuletzt aufgerufen am:	18. Dezember 2022

Weiterführende Quellen

- Neural Networks for Machine Learning - Overview of mini-batch gradient descent
- Adaptive Subgradient Methods for Online Learning and Stochastic Optimization
- No More Pesky Learning Rates
- ADADELTA: An Adaptive Learning Rate Method
- A fast natural Newton method
- Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32
- Natural Gradient Works Efficiently in Learning