

Gradientenverfahren

Wie kann man Minima einer differenzierbaren Abbildung $f : \mathbb{R}^n \rightarrow \mathbb{R}$ finden?

Gradientenverfahren

- An jedem Punkt $x_k \in \mathbb{R}^n$ zeigt der negative Gradient $d_k := -\nabla f(x_k)$ in die steilste Abstiegsrichtung.
- Für hinreichend kleines α_k folgt mit Satz über die lokale Linearisierung:
$$f(x_{k+1}) = f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k df(x_k)d_k + R(\alpha_k dk)$$
- Setze $x_{k+1} = x_k + \alpha_k d_k$
- Es gilt $f(x_{k+1}) \leq f(x_k)$, falls $\nabla f(x_k) \neq 0$
- Falls die Folge $f(x_k)$ beschränkt ist, so ist dieser Fixpunkt x^* ein Minimum, da $\nabla f(x^*) = 0$ gelten muss.

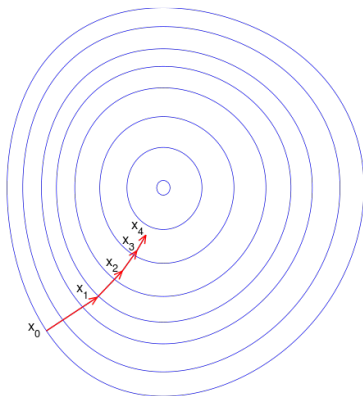


Figure: Quelle: Wikipedia

Höhenlinien

Sei $f : \mathbb{R}^n \rightarrow \mathbb{R}$ eine differenzierbare Funktion. Eine Kurve $\gamma : I \rightarrow \mathbb{R}^n$, auf der f konstant ist, also $f(\gamma(t)) = c$ für ein festes $c \in \mathbb{R}$ gilt, heißt Höhenlinie.

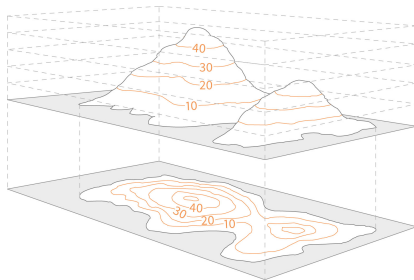


Figure: Quelle:

<https://getoutside.ordnancesurvey.co.uk/guides/understanding-map-contour-lines-for-beginners/>

Höhenlinien

Der Gradient steht senkrecht auf Höhenlinien.

Backpropagation

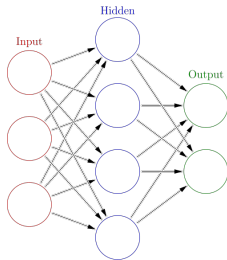
Das Gradientenverfahren angewendet auf eine Lossfunktion eines neuronalen Netzes wird als Backpropagation bezeichnet. Gegeben ist ein neuronales Netz $f : \Omega \times \mathbb{R}^n \rightarrow \mathbb{R}^m$, und ein Datensatz $D := \{(x_i, y_i)\}$ mit $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}^m$. Finde Gewichte Ω , so dass Lossfunktion

$$L_D : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}$$

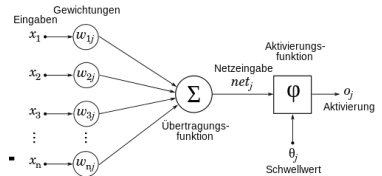
minimal wird. Zum Beispiel

$$L_D(\omega) := \sum_{(x_i, y_i) \in D} (f(\omega, x_i) - y_i)^2$$

.



Figure



Figure

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- While $\|\nabla L_D(\omega)\| > \epsilon$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- While $\|\nabla L_D(\omega)\| > \epsilon$

- Bestimme α_k mit

$$L_D(\omega_k + \alpha d_k) = L_D(\omega_k) + \alpha_k dL_D(\omega_k) d_k + R(\alpha_k dk)$$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- While $\|\nabla L_D(\omega)\| > \epsilon$
- Bestimme α_k mit
$$L_D(\omega_k + \alpha d_k) = L_D(\omega_k) + \alpha_k dL_D(\omega_k) d_k + R(\alpha_k dk)$$
- Setze $\omega_{k+1} := \omega_k + \alpha_k d_k$.

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- While $\|\nabla L_D(\omega)\| > \epsilon$
- Bestimme α_k mit
$$L_D(\omega_k + \alpha d_k) = L_D(\omega_k) + \alpha_k dL_D(\omega_k) d_k + R(\alpha_k dk)$$
- Setze $\omega_{k+1} := \omega_k + \alpha_k d_k$.
- $k \leftarrow k + 1$

Mini Batch

- Datensatz D sehr groß (Big Data)

Mini Batch

- Datensatz D sehr groß (Big Data)
- Berechnung des Gradienten der Lossfunktion entsprechend aufwendig.

Mini Batch

- Datensatz D sehr groß (Big Data)
- Berechnung des Gradienten der Lossfunktion entsprechend aufwendig.
- Wende Backpropagation auf Teilräume $D' \subset D$ an (Minibatch).

Mini Batch

- Datensatz D sehr groß (Big Data)
- Berechnung des Gradienten der Lossfunktion entsprechend aufwendig.
- Wende Backpropagation auf Teilräume $D' \subset D$ an (Minibatch).
- $\#D' = 1$ stochastischer Gradientenabstieg.

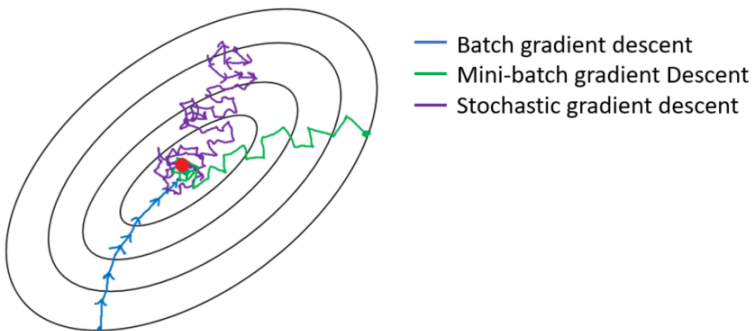


Figure: Quelle: <https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a>

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- Wähle Teilmenge $D'_0 \subset D$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- Wähle Teilmenge $D'_0 \subset D$
- While $\|\nabla L_{D'_k}(\omega)\| > \epsilon$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- Wähle Teilmenge $D'_0 \subset D$
- While $\|\nabla L_{D'_k}(\omega)\| > \epsilon$

- Bestimme α_k mit

$$L_{D'_k}(\omega_k + \alpha d_k) = L_{D'_k}(\omega_k) + \alpha_k dL_{D'_k}(\omega_k) d_k + R(\alpha_k dk)$$

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- Wähle Teilmenge $D'_0 \subset D$
- While $\|\nabla L_{D'_k}(\omega)\| > \epsilon$
- Bestimme α_k mit
$$L_{D'_k}(\omega_k + \alpha d_k) = L_{D'_k}(\omega_k) + \alpha_k dL_{D'_k}(\omega_k) d_k + R(\alpha_k dk)$$
- Setze $\omega_{k+1} := \omega_k + \alpha_k d_k$.

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- Wähle Teilmenge $D'_0 \subset D$
- While $\|\nabla L_{D'_k}(\omega)\| > \epsilon$
- Bestimme α_k mit
$$L_{D'_k}(\omega_k + \alpha d_k) = L_{D'_k}(\omega_k) + \alpha_k dL_{D'_k}(\omega_k) d_k + R(\alpha_k dk)$$
- Setze $\omega_{k+1} := \omega_k + \alpha_k d_k$.
- Wähle neue Teilmenge $D'_{k+1} \subset D$.

Backpropagation

- Initialisiere $k := 0$ und zufällige Gewichte w_0 .
- Initialisiere Genauigkeit $\epsilon > 0$
- Wähle Teilmenge $D'_0 \subset D$
- While $\|\nabla L_{D'_k}(\omega)\| > \epsilon$
- Bestimme α_k mit
$$L_{D'_k}(\omega_k + \alpha d_k) = L_{D'_k}(\omega_k) + \alpha_k dL_{D'_k}(\omega_k) d_k + R(\alpha_k dk)$$
- Setze $\omega_{k+1} := \omega_k + \alpha_k d_k$.
- Wähle neue Teilmenge $D'_{k+1} \subset D$.
- $k \leftarrow k + 1$

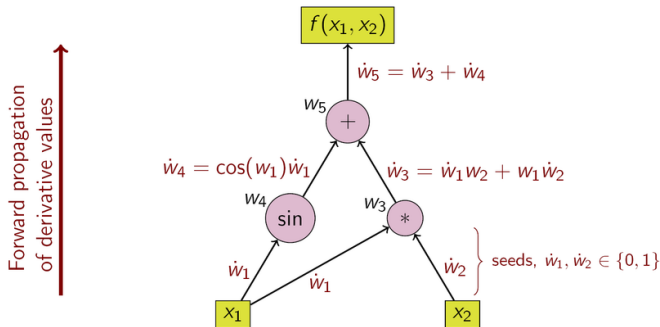


Figure: Quelle: Wikipedia

Automatisches Ableiten in Pytorch
Automatisches Ableiten in JAX

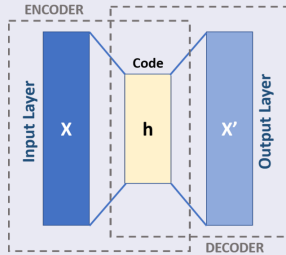
Diese Person existiert nicht!

<https://this-person-does-not-exist.com/de>

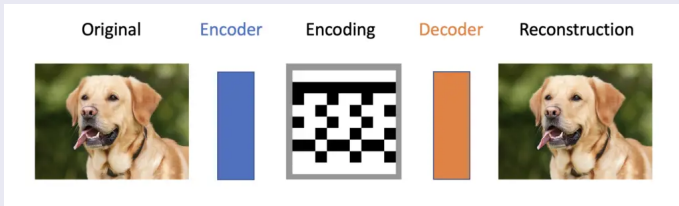


Generative Modelle

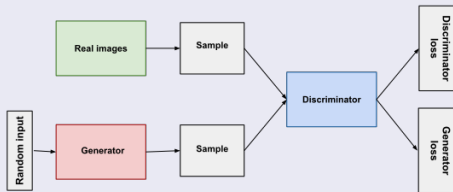
Autoencoder



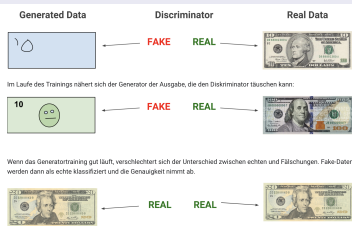
Autoencoder



GAN Architektur

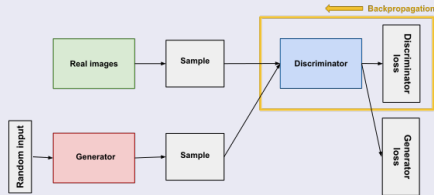


Generator & Discriminator



Generative Modelle

Training Discriminator



Training Generator

