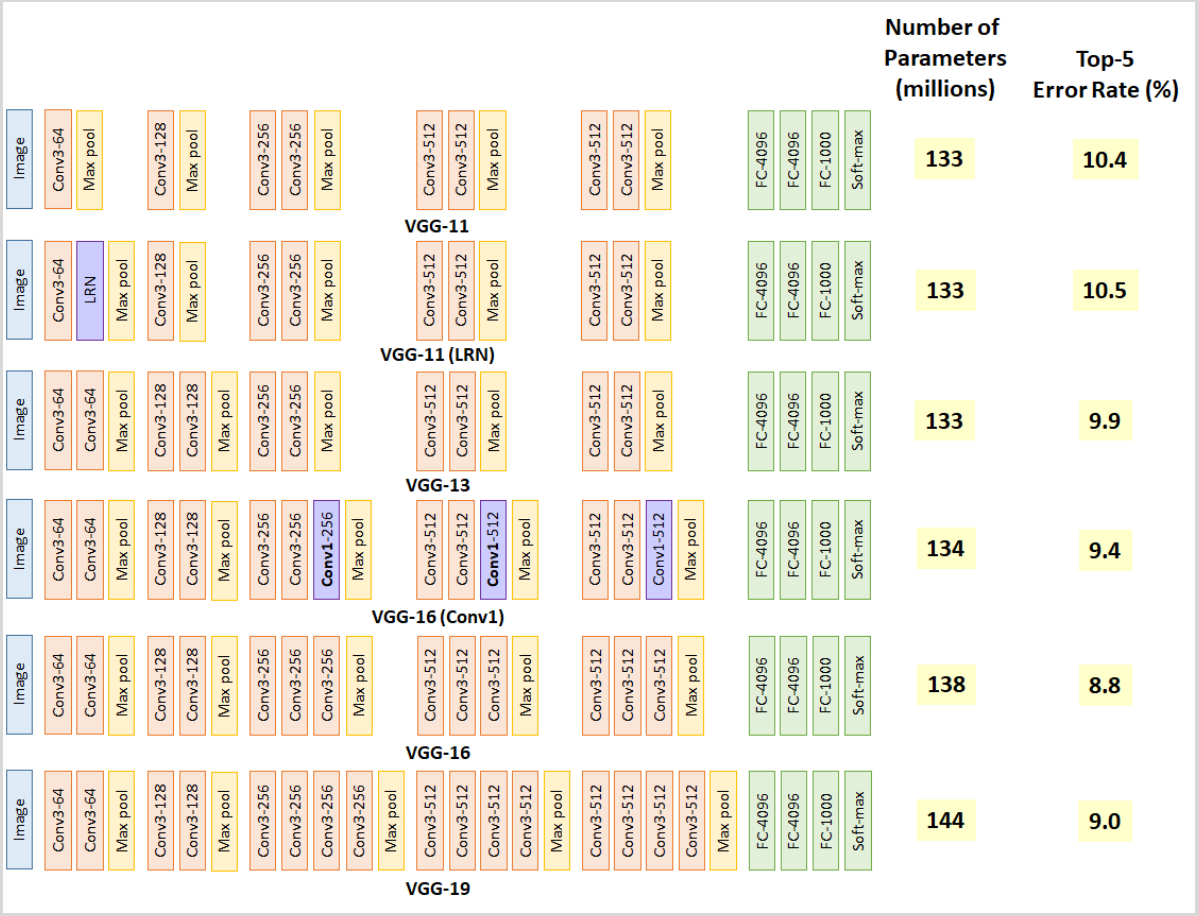
Deep Neural Networks

KI Q&A der AG5

Disclaimer

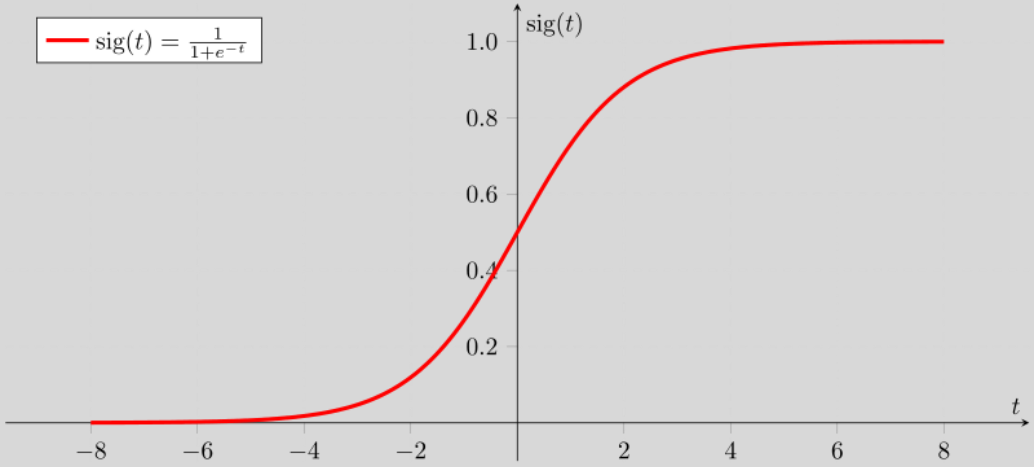
Tiefe Neuronale Netze

- Komplexe Aufgaben brauchen tiefere Netze
  - VGG-19: 20 Layer (133 Millionen Parameter)
  - GPT-3: 96 Layer (175 Billionen Parameter)



Probleme

- Exploding Gradients
  - Beim Propagieren der Fehler wird dieser immer größer
  - Exponentieller Anstieg
  - Dadurch große Schritte
    → Oszillation und „Überspringen“ der Minima
- Vanishing Gradients
  - Beim Propagieren der Fehler wird dieser immer kleiner
  - Exponentiell fallend
  - Dadurch kleine oder gar keine Schritte
    → Langsames oder gar kein Lernen



Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \qquad \text{(BP3)}$$

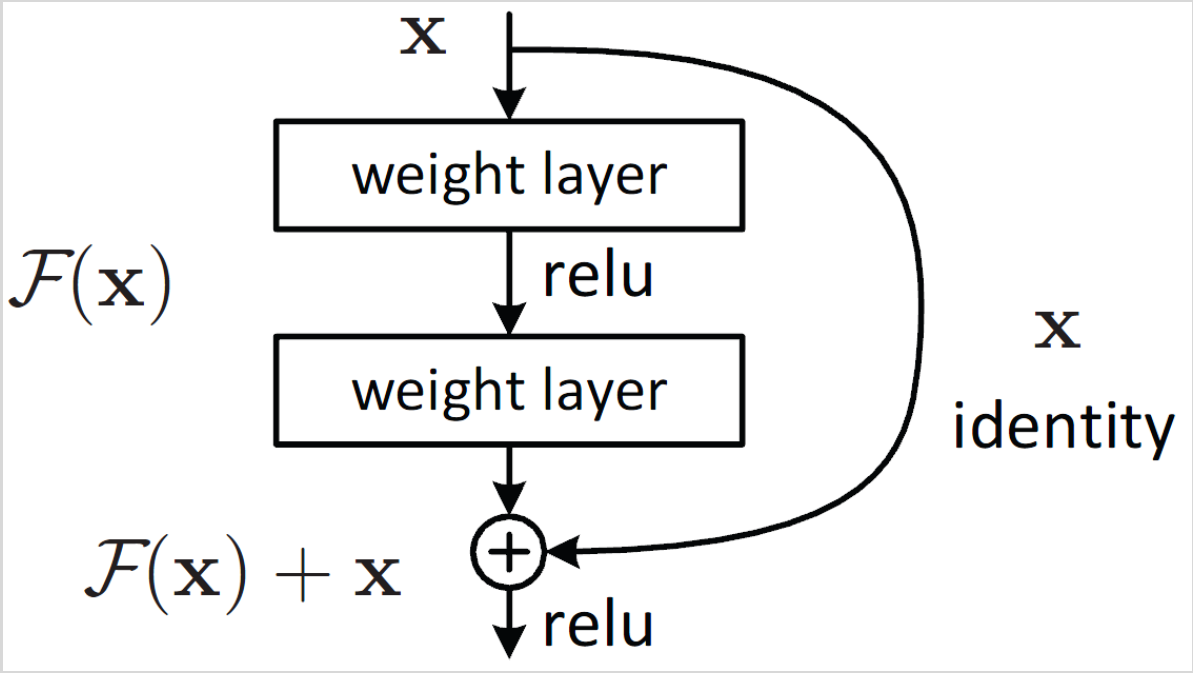
$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad \text{(BP4)}$$

Lösungen

- Nicht-saturierende Aktivierungsfunktion (gegen Vanishing Gradient)
  - ReLU
  - LeakyReLU
- Gute Initialisierung der Gewichte
  - Z.B. Glorot Initialisierung
- Gradient Clipping
  - Schneide Gradient bei 1 oder bei 0 ab
- Oder…

Das Netz lernen lassen, wie viel Layer es braucht

- ResNet mit Skip Connections
- Wir addieren den Input eines ResBlocks auf dessen Output
- ResBlöcke können übersprungen werden
  - Gewichte in Blöcken können 0 werden
  - Dann werden Layer ignoriert
  - Gradient kann Block „umfließen“
- Das Netz lernt nur so viele Layer zu verwenden, wie es braucht
- Weiterer Vorteil: Spätere Layer können unverarbeitete Information nutzen



Wie bauen wir eine Addition ein?

- Lösung: Tensorflow Functional API
- Alternative zu Sequential Model
- Wir geben ein Netz als Aufruf von Funktionen an

BatchNormalization

- Prinzip
  - Nehme den Output eines Layers für den aktuellen Batch
  - Normalisiere ihn auf eine Normalverteilung
    - Mit lernbarer Varianz
    - Und lernbarem Mittelwert
- Resultat: Schnelleres und besseres Lernen
- Niemand weiß wirklich genau warum BatchNormalization funktioniert
  - Momentan beste Idee: Weniger Abhängigkeiten zwischen Ebenen
  - Oder/Und: Weniger lokale Optima

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Aufgabe 1

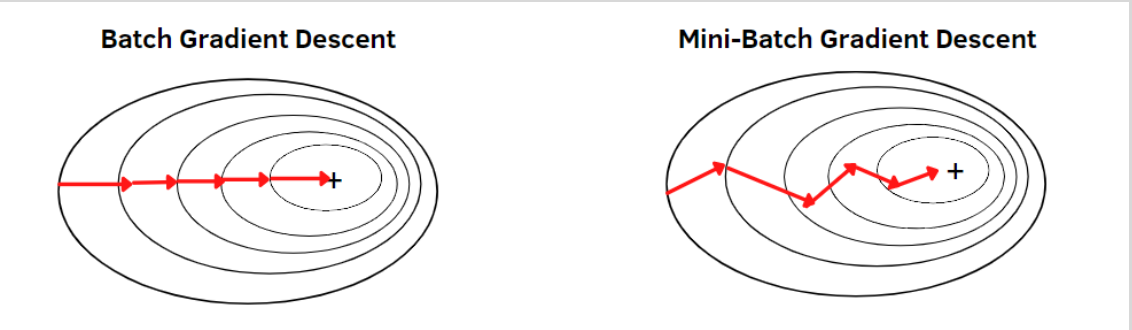
https://t1p.de/QA-Session-4-Aufgabe-1

Noch ein Problem tiefer Netze

- Tiefe Netze brauchen sehr lange, um zu lernen
- Wir wollen das lernen beschleunigen
- Lösung: Anpassungen für den Lernalgorithmus

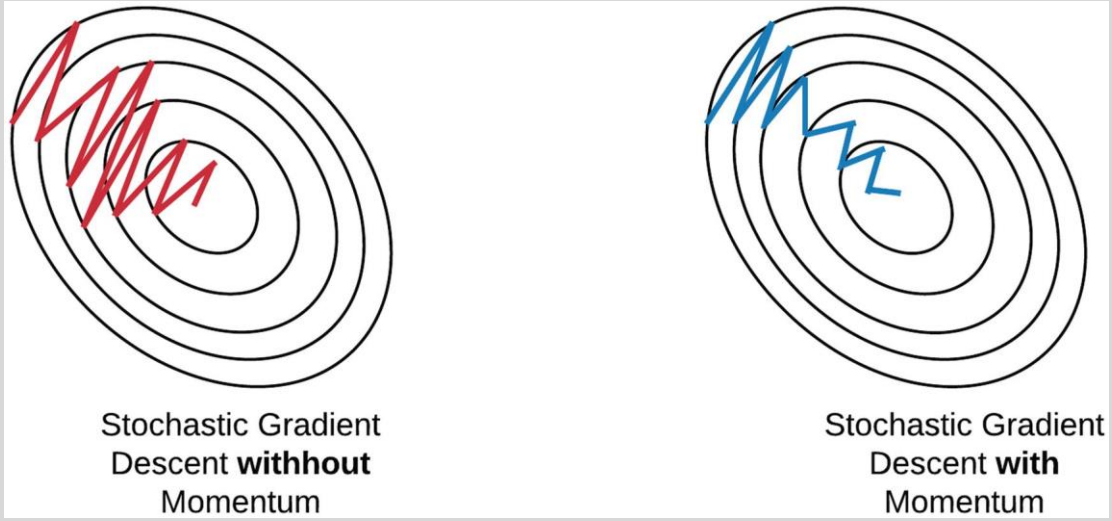
(Stochastic) Gradient Descent

- Gradient Descent
  - Berechne die Fehler für alle Datenpunkte
  - Passe die Gewichte an, so dass dieser reduziert wird
  - Probleme:
    - Dauert lange, da viele Fehler berechnet werden müssen
    - Viele Iterationen nötig bis Minimum erreicht wurde
- Stochastic Gradient Descent (SGD)
  - Berechne den Fehler für einen Teil der Datenpunkte
  - Passe die Gewichte an, so dass dieser reduziert wird
  - Vorteil:
    - Mehr Updates in jeder Epoche
    - Dadurch schnellere Konvergenz



Momentum

- Problem: Stochastic Gradient Descent hat viele Oszillationen
- Lösung: Momentum
  - Wir fügen dem Update einen Teil des letzten Updates hinzu
- Falls unterschiedliche Richtungen – langsamer
- Falls gleiche Richtungen – schneller
- Also Konzentration auf Hauptrichtung



AdaGrad

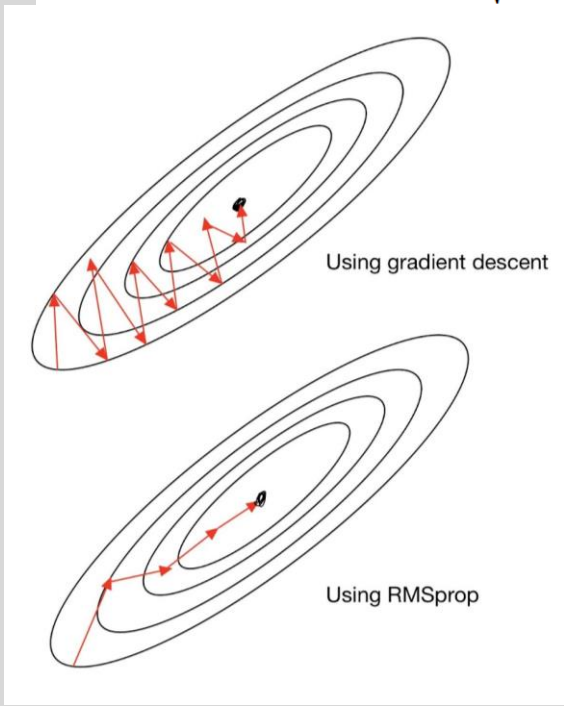
- Optimiert eine Lernrate für jede Dimension
- Parameter, die wenige Updates erhalten kriegen eine niedrige Lernrate
- Parameter, die viele Updates erhalten kriegen eine hohe Lernrate
- Vorteil:
  - Bestenfalls geringerer Einfluss der initialen Lernrate
- Nachteil:
  - Lernrate für eine Dimension kann gegen null konvergieren

RMSProp

- Dividiere die Lernrate für jeden Parameter durch die Wurzel des Mittleren Quadrats aller vorherigen Gradienten
- Dadurch passt sich Lernrate automatisch an
- Da Gradienten Summe steigt wird Lernrate im Lauf des Trainings kleiner
  → Anfangs große Schritte dann immer kleinere
- Oszillationen werden gedämpft
- Wir können eine größere initiale Lernrate wählen

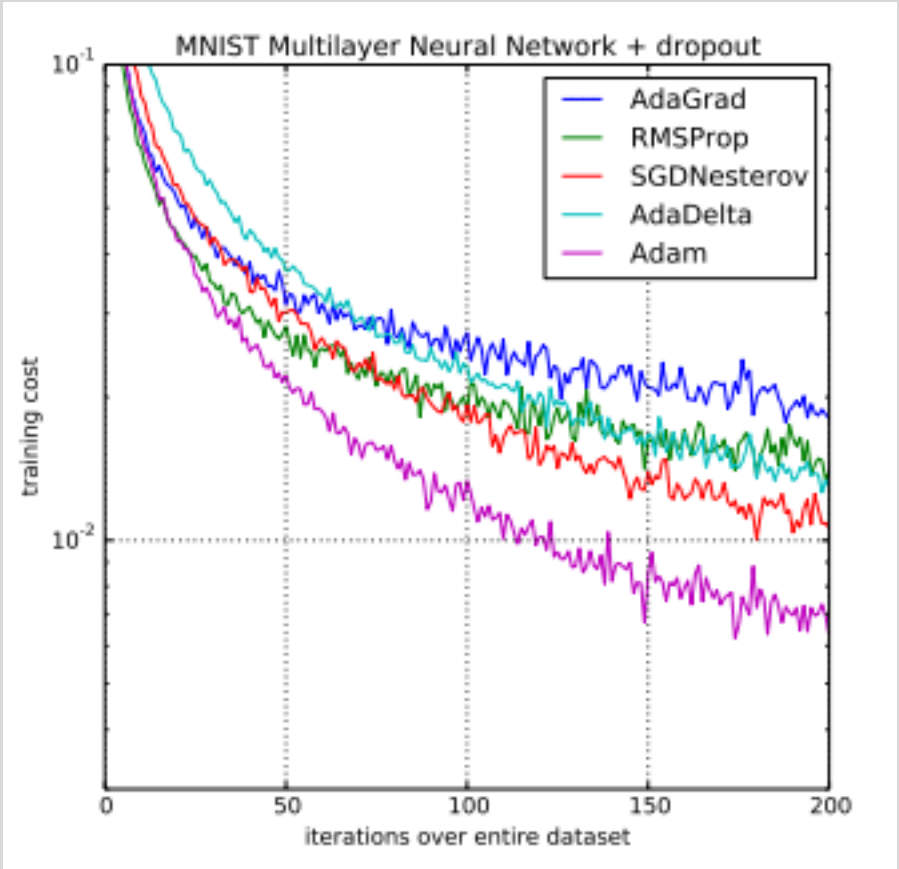
$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$



Adam

- Kombiniert Momentum mit RMSProp
- Verhinderung von Oszillationen durch Teilen durch Wurzel des mittleren Quadrats
- Konzentration auf eine Richtung durch Momentum
- Zusätzlich: Bias Korrektur

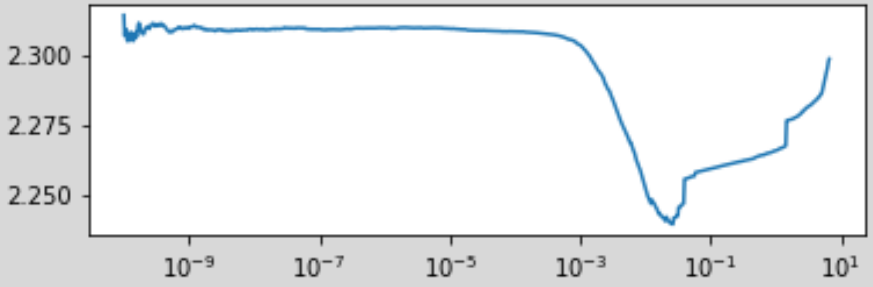


Lernrate

- Der einzige Parameter, der normalerweise getunt wird ist die initiale Lernrate
- Kann großen Einfluss auf das Ergebnis haben
- Zu kleine Schritte:
  - Langes Training
  - Feststecken in lokalen Minima
- Zu große Schritte
  - Langes Training
  - Verfehlen des globalen Minimums

Lernratentrick

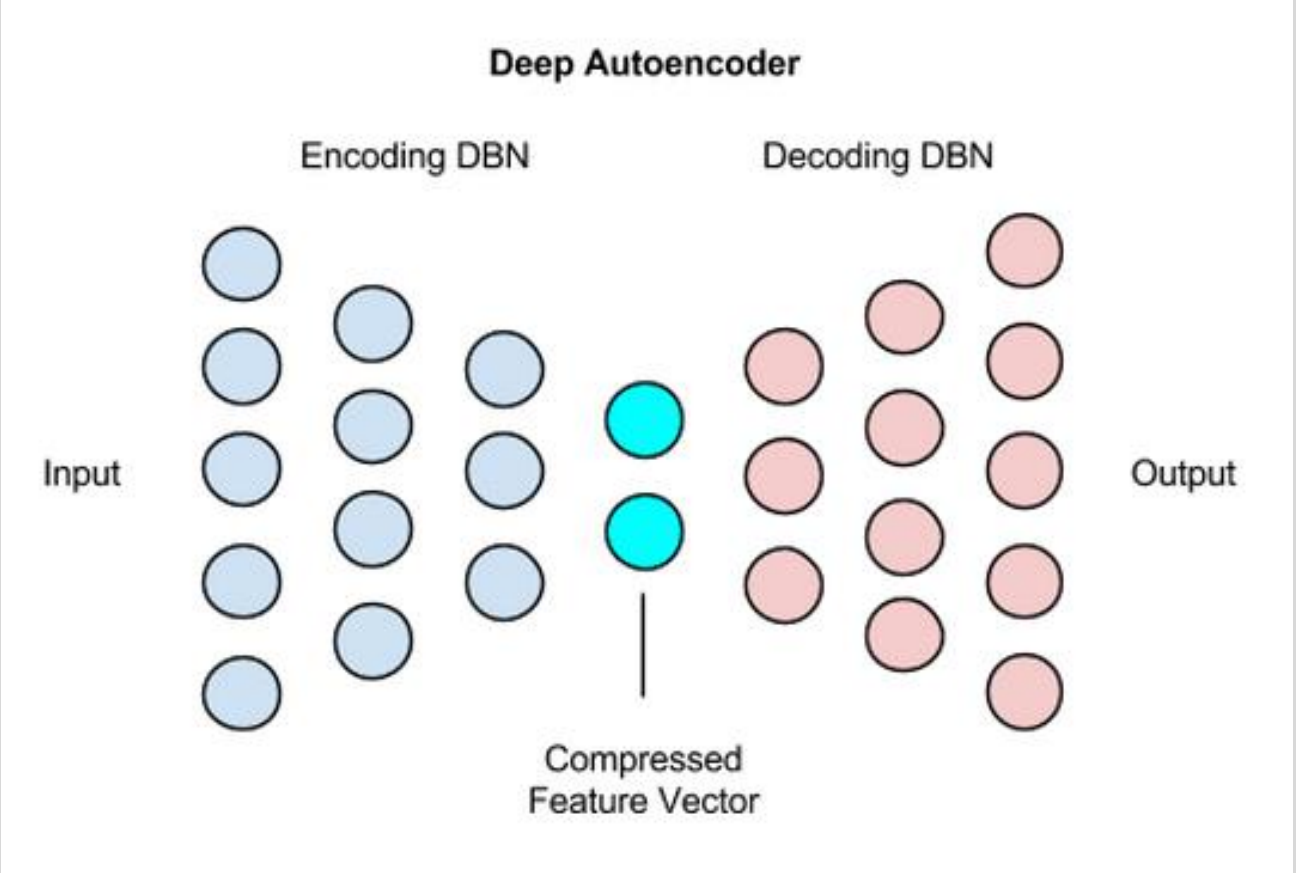
- Eine Möglichkeit, eine möglichst optimale Lernrate zu bestimmen
- Starte mit sehr kleiner Lernrate
- Nach jedem Batch: Erhöhe die Lernrate exponentiell
- Plotte die Veränderung der Loss-Funktion gegen die Lernrate in dieser Iteration
- Wähle die Lernrate, bei der der schnellste Abstieg zu erkennen ist



Aufgabe 2

https://t1p.de/QA-Session-4-Aufgabe-2

Deep Autoencoder



Deep Denoising Autoencoder

- Input ist verrauschtes Bild, Target ist rauschfreies Bild
- In der Mitte muss die Information „vergessen“ werden, die Rauschen ist

Sparse Autoencoder

- Wir fügen einen Constraint in der Mitte ein
  - Möglichst viele Nuller
  - Z.B. über L1 Norm
- Dadurch Automatisches One-Hot-Encoding
  - Entspricht Klassifikation

Erkennung von Outliern

- Outlier zeichnen sich dadurch aus, dass Sie anders sind als die Daten im Trainingsdatensatz
- Einen Outlier kann man also dadurch erkennen, dass er weniger gut vom Autoencoder rekonstruiert werden kann
- Daher ist die Accuracy besonders niedrig
- Kriterium was ein Outlier ist (welche Accuracy niedrig ist) muss aber selbst festgelegt werden

Aufgabe 3

https://t1p.de/QA-Session-4-Aufgabe-3