Aufgabe 21 (Variational Autoencoders[1])

(a) Die "evidence lower bound" (ELBO) für Wahrscheinlichkeitsdichten p und q ist definiert als

$$\mathcal{L}_{p,q}^{\mathsf{ELBO}}(x) = \int \big(\log p(x,z) - \log q(z|x)\big) q(z|x)\,\mathrm{d}z.$$

Verifizieren Sie die beiden Relationen

$$-\mathcal{L}_{p,q}^{\mathsf{ELBO}}(x) + \log p(x) = D_{\mathsf{KL}}\left(q(z|x) \parallel p(z|x)\right)$$

und

$$\mathcal{L}_{p,q}^{\mathsf{ELBO}}(x) = -D_{\mathsf{KL}}\left(q(z|x) \parallel p(z)\right) + \int \log p(x|z) q(z|x)\,\mathrm{d}z.$$

[3 Punkte]

Hinweis: Sie dürfen annehmen, dass alle (bedingten) Wahrscheinlichkeitsdichten positiv sind, d.h. Spezialfälle (wie Division durch Null) brauchen nicht berücksichtigt werden.

(b) Die Kullback-Leibler-Divergenz zwischen zwei *Normalverteilungen* lässt sich analytisch berechnen: es sei

$$f_{\mu,\sigma} : \mathbb{R} \to \mathbb{R}, \quad f_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

die Dichte der Normalverteilung mit Mittelwert μ und Standardabweichung σ. (Der Einfachheit halber betrachten wir hier eindimensionale Verteilungen; das Resultat lässt sich auch auf höhere Dimensionen verallgemeinern.) Zeigen Sie: für $q = f_{\mu,\sigma}$ und $p = f_{0,1}$ gilt

$$D_{\mathsf{KL}}\left(q \parallel p\right) = -\frac{1}{2}\left(1 + \log(\sigma^2) - \mu^2 - \sigma^2\right).$$

[3 Punkte]

Aufgabe 22 (Generative Adversarial Network angewendet auf MNIST)

Generative Adversarial Networks (GANs)[2] sind eine Klasse generativer Modelle, die – ausgehend von einer Zufallsstichprobe $x^{(1)}, x^{(2)}, \ldots$ einer (unbekannten) Wahrscheinlichkeitsdichte p_{exact} – weitere Realisierungen (Samples) generieren, *ohne* ein Wahrscheinlichkeitsmodell explizit aufzustellen. Die $x^{(i)}$ könnten z.B. Pixelbilder menschlicher Gesichter sein, so dass das Netzwerk nach dem Trainieren künstliche Gesichter erzeugt[3]. Ein GAN besteht aus zwei Gegenspielern: der "Generator" G versucht, möglichst glaubwürdige Realisierungen zu erzeugen, während der "Diskriminator" D echte Stichproben von generierten Realisierungen unterscheiden soll. Konkret ist $D(x)$ eine Zahl aus dem Intervall $[0,1]$, interpretiert als Wahrscheinlichkeit, dass die Stichprobe x echt ist. G erzeugt eine Realisierung aus einem Zufallsvektor z (der z.B. bei Gesichtsgenerierung abstrakte Charakterisierungen wie Haarfarbe, Brille vorhanden?, ... enthalten könnte). Mathematisch ausgedrückt gelangt man zur Min-Max-Formulierung

$$\min_G \max_D V(D,G)$$

wobei

$$V(D,G) = \mathbb{E}_{x\sim p_{\mathsf{exact}}}[\log(D(x))] + \mathbb{E}_{z\sim p_{\mathsf{noise}}}[\log(1 - D(G(z)))].$$

Hierbei bezeichnet $\mathbb{E}$ den stochastischen Erwartungswert, und $\mathbb{E}_{x\sim p}[f(x)] = \int f(x)p(x)\,\mathrm{d}x$. Sowohl D als auch G werden als künstliche neuronale Netze mit Parametern θ_d bzw. θ_g umgesetzt, die simultan mittels Backpropagation optimiert werden (Bezeichnung D_{θ_d} bzw. G_{θ_g}). Somit lautet ein Schritt des Gradientenverfahrens mit Lernrate η in der einfachsten Version:

$$\theta_d \to \theta_d + \eta \nabla_{\theta_d} V(D_{\theta_d}, G_{\theta_g}), \tag{1a}$$

$$\theta_g \to \theta_g - \eta \nabla_{\theta_g} V(D_{\theta_d}, G_{\theta_g}). \tag{1b}$$

[1] Diederik P. Kingma, Max Welling: Auto-encoding variational Bayes. 2014 (https://arxiv.org/abs/1312.6114)

[2] Ian J. Goodfellow et al.: Generative adversarial networks. 2014 (https://arxiv.org/abs/1406.2661)

[3] siehe etwa https://thispersondoesnotexist.com

Da der erste Term in $V(D,G)$ nicht von G abhängt, fällt er bei der Berechnung des Gradienten bezüglich θ_g heraus. In der Praxis wird der zweite Term in $V(D,G)$ beim Gradientenschritt für den *Generator* ersetzt durch den zu *maximierenden* Term

$$\tilde{V}_2(D,G) = \mathbb{E}_{z\sim p_{\text{noise}}}\left[\log(D(G(z)))\right],$$

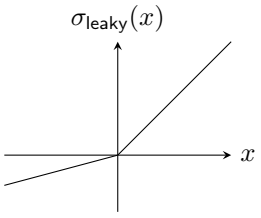
d.h. Gl. (1b) ersetzt durch

$$\theta_g \to \theta_g + \eta\nabla_{\theta_g}\tilde{V}_2(D_{\theta_d}, G_{\theta_g}). \tag{1b'}$$

Die Motivation hierfür ist ein möglichst großer Gradient im Fall $D(G(z)) \approx 0$, d.h. falls G nicht gut funktioniert.

In `gan_mnist.zip` auf der Webseite zur Vorlesung[4] findet sich die Vorlage einer GAN-Implementierung mittels TensorFlow zur Generierung "künstlicher" handschriftlicher Ziffern ausgehend vom MNIST Datensatz[5].

(a) Insbesondere für GANs stellt es sich als vorteilhaft heraus, die ReLU-Aktivierungsfunktion (die für negative Argumente inaktiv wird) durch die "Leaky ReLU" Aktivierungsfunktion mit Parameter α zu ersetzen:



$$\sigma_{\text{leaky}}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

Implementieren Sie diese Abbildung mittels TensorFlow innerhalb `leaky_relu(x, alpha)` in der Vorlage `gan_mnist.py`. [2 Punkte]

Hinweis: `tf.maximum` bzw. `tf.minimum` könnten hierfür nützlich sein.

(b) Machen Sie sich mit der Implementierung in `gan_mnist.py` vertraut, und erklären Sie kurz, wie das Zielfunktional V bzw. $\tilde{V}_2$ mittels TensorFlow umgesetzt wird. Lassen Sie nun das Programm laufen, um "künstliche" handschriftliche Ziffern zu erzeugen. [2 Punkte]

Hinweis: Das Programm speichert Beispiel-Realisierungen nach jeder Trainingsepoche ab.

(c) Die bisherige Architektur des Diskriminators und Generators besteht aus Dense Layers. Verbesserte Ergebnisse erhält man durch Berücksichtigung der räumlichen Struktur, also mittels Convolutional Layers. Ersetzen Sie hierzu die Zeile `mtype = 'dense'` durch `mtype = 'conv'` innerhalb `main()`, und lassen Sie das Programm nochmals laufen. Erklären Sie kurz den Unterschied zwischen der (vom Generator verwendeten) Conv2DTranspose Layer im Vergleich zur "konventionellen" Conv2D Layer. [2 Punkte]

Tutoraufgabe 11 (Reparameterization Trick)

Der "Reparameterization Trick" dient dazu, Modelle mit parameterabhängigen Wahrscheinlichkeitsverteilungen bezüglich dieser Parameter zu differenzieren. Es bezeichne p_θ eine Wahrscheinlichkeitsdichte mit Parameter θ, z.B. die Normalverteilung $p_\theta = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$. Falls lediglich Samples $x^{(i)} \sim p_\theta$ zur Verfügung stehen, dann ist a priori nicht klar, wie der Gradient bezüglich θ bestimmt werden kann. Als konkretes Szenario betrachten wir den Erwartungswert

$$\mathbb{E}_{x\sim p_\theta}[f(x)] = \int f(x)p_\theta(x)\,\mathrm{d}x$$

und nehmen an, dass das Integral durch Monte-Carlo-Integration approximiert wird:

$$\mathbb{E}_{x\sim p_\theta}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f\big(x^{(i)}\big), \qquad x^{(i)} \sim p_\theta.$$

Eine naheliegende Idee zur Berechnung des Gradienten ist die Darstellung

$$\nabla_\theta\,\mathbb{E}_{x\sim p_\theta}[f(x)] = \int f(x)\big(\nabla_\theta \log p_\theta(x)\big)p_\theta(x)\,\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} f\big(x^{(i)}\big)\left(\nabla_\theta \log p_\theta\big(x^{(i)}\big)\right), \quad x^{(i)} \sim p_\theta,$$

allerdings ist dies in der Praxis oft zu ungenau. Stattdessen verwendet der Reparameterization Trick eine deterministische Abbildung g_θ und eine (von θ unabhängige) Wahrscheinlichkeitsverteilung q, so dass das reparametrisierte

$$x = g_\theta(\epsilon), \quad \epsilon \sim q$$

weiterhin die Verteilung $x \sim p_\theta$ besitzt. Z.B. für die obige Normalverteilung bietet sich $g_\theta(\epsilon) = \mu_\theta + \epsilon\sigma_\theta$ mit $\epsilon \sim \mathcal{N}(0,1)$ an. Hiermit lässt sich der Erwartungswert umschreiben als

$$\mathbb{E}_{x\sim p_\theta}[f(x)] = \int f(g_\theta(\epsilon))q(\epsilon)\,\mathrm{d}\epsilon \approx \frac{1}{N}\sum_{i=1}^{N} f\big(g_\theta\big(\epsilon^{(i)}\big)\big), \qquad \epsilon^{(i)} \sim q.$$

Wie lautet also der Gradient bezüglich θ nach Reparametrisierung? Leiten Sie außerdem eine Beziehung zwischen p_θ, g_θ und q her. Wie können (in einer Dimension) g_θ und q bei bekannter Verteilungsfunktion von x gewählt werden?

[4] https://tu-dresden.de/mn/math/wir/mendl/studium/courses/mosim_2019_sose

[5] basierend auf http://cs231n.github.io/assignments2018/assignment3