Exercise Sheet	02	**1. Team Partner:**	Lennart Slusny
Task	01	**2. Team Partner:**	Joschka Strüber

(a) How can LSTMs and other gated memory units learn to count and solve other discrete tasks? What is expected to happen during training?

Answer:

The time steps of an RNN act as a natural source of discreteness. However, because of the aforementioned smooth and continuous functions make sharp calculations in ungated RNNs very difficult to learn.

Here comes the gates and especially the forget gate into play. The cell state s_c^t is computed as the input gate activation times the squashed cell input added to the forget gate multiplied with the previous cell state:

$$s_c^t = x_l^t \odot g_c(\text{net}_c^t) + x_\phi^t \odot s_c^{t-1} \tag{1}$$

While the sigmoid activation function for the gates is smooth and continuous in theory, in practice the sigmoid saturates very quickly to almost 0 for small inputs and almost 1 for large inputs. As a result, the cell state is very often not a truly continuous mixture between inputs and recurrences, but rather the switches turn completely on or off. This drastic behavior of the gates allows to learn sharp and pretty much discrete function.

This effect is also very beneficial for training these networks and allows to effectively learn much longer dependencies. Whenever the values of the forget gate are almost 1, the gradients backpropagating through time from $\nabla_\theta s_c^t$ to $\nabla_\theta s_c^{t-1}$ remain the same. Once the gate is turned off and its values are close to zero, the corresponding gradient is cut off between time steps.

(b) Consider a single LSTM unit with D memory cells. Starting with

$$\delta_\phi^t = \frac{\partial E}{\partial \text{net}_\phi^t} \tag{2}$$

derive the forget gate gradient of the LSTM:

$$\delta_\phi^t = \varphi_\phi'(\text{net}_\phi^t) \sum_{c=1}^{D} \zeta_c^t s_c^{t-1} \tag{3}$$

Answer:

$$\begin{aligned}
\delta_\phi^t &= \frac{\partial E}{\partial \text{net}_\phi^t} && (4)\\
&= \frac{\partial x_\phi^t}{\partial \text{net}_\phi^t} \frac{\partial E}{\partial x_\phi^t} && (5)\\
&= \frac{\partial \varphi_\phi(\text{net}_\phi^t)}{\partial \text{net}_\phi^t} \frac{\partial E}{\partial x_\phi^t} && (6)\\
&= \varphi_\phi'(\text{net}_\phi^t) \frac{\partial E}{\partial x_\phi^t} && (7)
\end{aligned}$$

(8)

This way, we derived the first part of the forget gate gradient. Next, we focus on the second part. The forget gate is used to compute the cell state for every of the D cells by element-wise multiplication. Hence, we have to consider the gradient for the vector of internal states at time step t:

$$\delta_\phi^t = \varphi_\phi'(\text{net}_\phi^t)\frac{\partial E}{\partial x_\phi^t} \tag{9}$$

$$= \varphi_\phi'(\text{net}_\phi^t)(\nabla_{x_\phi^t} s^t)^T \nabla_{s^t} E \tag{10}$$

$$= \varphi_\phi'(\text{net}_\phi^t)\sum_{c\in D}\frac{\partial s_c^t}{\partial x_\phi^t}\frac{\partial E}{\partial s_c^t} \tag{11}$$

$$= \varphi_\phi'(\text{net}_\phi^t)\sum_{c\in D}\frac{\partial\left(x_l^t g_c(\text{net}_c^t) + x_\phi^t s_c^{t-1}\right)}{\partial x_\phi^t}\zeta_c^t \tag{12}$$

$$= \varphi_\phi'(\text{net}_\phi^t)\sum_{c\in D} s_c^{t-1}\zeta_c^t \tag{13}$$

(c) Can bidirectional RNNs be effectively used for online classification, that is generating a classification label for every new input immediately? Explain briefly.

Answer: No, that is in general not possible. To make inference, a bidirectional RNN needs both pretemporal and posttemporal context to compute an output. However, in online classification, only the pretemporal context is available. To get the posttemporal context, we would have to get to the end of a timeseries and compute the recurrence back in time, which is impossible in this context.

Task 02

(a) Name three forms of commonly applied regularization techniques for neural networks and describe their effects.

Answer:

Early stopping uses a separate evaluation set to monitor the error of the model. At one point, the model will very likely stop to generalize well and start to overfit on the training set. Since we monitor the error on the validation set, that is not used for training, we can very likely find this moment of the training process, stop the training and return the model that performed best on the validation set. Because it is assumed to come from the same distribution as the test set, we expect the returned model to be the best of that training regime on the test set as well.

Weight decay adds a regularization term based on the model's weight to the error:

$$\hat{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda\Omega(\mathbf{w}) \tag{14}$$

Ω can be any metric that punishes large weights, e.g. the Manhattan or Euclidean distance. Since the regularization is part of the loss, we also calculate the gradient with respect to the regularization that is used for updating the weights:

$$w_i \leftarrow w_i + \eta \frac{\partial E(\mathbf{w})}{\partial w_i} + \eta\lambda \frac{\partial E(\mathbf{w})}{\partial w_i} \tag{15}$$

In other words, the weights are decreased or "decay" in each training step based on their size, hence the method's name.

Deep neural network tend to produce sharp, low-entropy, over-confident results by memorizing the training data. To counteract this effect, label smoothing adds noise to the training labels, by decreasing the values of positive classes and increase those of negative classes in one-hot vectors. Given a single-label classification problem with N classes, we choose a small ϵ as hyper parameter and modify the labels y as follows:

$$\hat{y}_i = \frac{\epsilon}{N-1} \qquad \text{, if } y_i = 0 \tag{16}$$

$$\hat{y}_i = 1 - \epsilon \qquad \text{, if } y_i = 1 \tag{17}$$

(b) Explain the internal covariate shift and how it has been addressed in the literature.

Answer:

Given a layer, when the parameters of the previous layer change, so does the statistics of the current's inputs. Therefore, the current layer has to adjust to changing targets during the training process and becomes instable, if the learning rates are too large or the weight initialization is chosen unfavorably.

This problem is addressed by batch normalization. This technique computes the mean and variance of each mini-batch and uses them to normalize a layer's input. This way, the input always has comparable statistics. Apart of the normalization, the input is also scaled and shifted by two learnable parameters γ and β.

At the test phase, the learnable parameters are frozen and the normalizing mean and variance parameters are computed over the whole data set as a moving average, to get a deterministic inference that works well for the whole training data set.

(c) Consider the following statement: The more parameters a neural network has the better it will generalize. Is this statement true?

Answer:

In general, a model with more parameters has a higher chance of learning more complex functions and has the ability to generalize well. However, in practice there is a higher chance of overfitting to the seen training data, which hampers the model's ability to generalize. This corresponds to the classical regime of learning theory with a training risk that decreases with increasing model capacity - and hence number of parameters in our case - and increasing test risk, if the model capacity becomes too large.

However, [1] have shown that the test error start decreasing again for models with a very large model capacity, eg. deep neural networks with millions of parameters. Cf. the visualization in figure 1. The left-hand side shows the classic bias-variance tradeoff, where the model start overfitting if the capacity becomes too large. The right side shows the double-descent risk-curve, where the test error suddenly starts decreasing again for even larger model capacities.

(d) What is the effect of residual blocks on the gradient flow in deep networks?

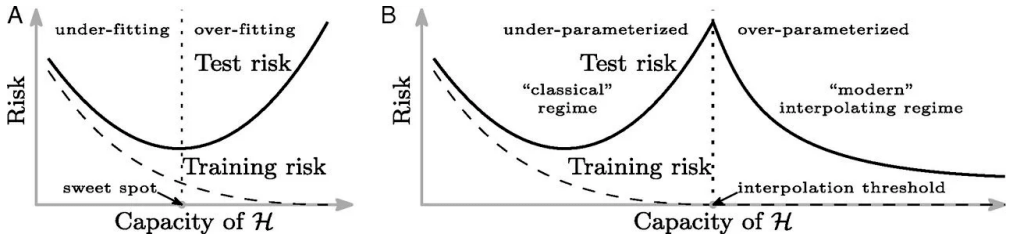

Figure 1: Curves for the training and test risk in the classical and modern regime of learning theory [1].

Answer:

Residual nets use shortcuts to take a layers output as identity and add them as an additional output by skipping a block of weight layers. This way, the function learned inside the residual block that was skipped can be seen as learning the offset from the identity.

Given a net activation x_l, a net activation x_L further down in the computation graph and weights functions f_i for the intermediate blocks that were skipped with residual connections, x_L can be computed based on x_l in the following way:

$$x_L = x_l + \sum_{i=l}^{L-1} f_i(x_i) \tag{18}$$

This makes sense, because the output after each residual block is added to the identity that is propagated via skip connections, but also further processed by the next residual blocks.

As a result, the gradient also backpropagates nicely as the identity, because of the skip connections, and it is modified additively based on the gradients of the residual blocks:

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}\frac{\partial x_L}{\partial x_l} \tag{19}$$

$$= \frac{\partial E}{\partial x_L}\left(1 - \frac{\partial}{\delta x_l}\sum_{i=l}^{L-1} f_i(x_i)\right) \tag{20}$$

In this way, the gradient is very unlikely to vanish, because of the identity skip connections. This decreases the chance of the vanishing gradient problem significantly and allows to train much deeper networks than before.

Task 03

(a) What is the general purpose of autoencoders (AEs) and what are their main components? Give an illustration.

Answer:

Autoencoders are based on a decoder and an encoder, as can be seen in figure 2. The encoder maps the input to a lower-dimensional latent space. The decoder tries to reproduce the input based on the latent state by minimizing a reconstruction error, eg. L2.

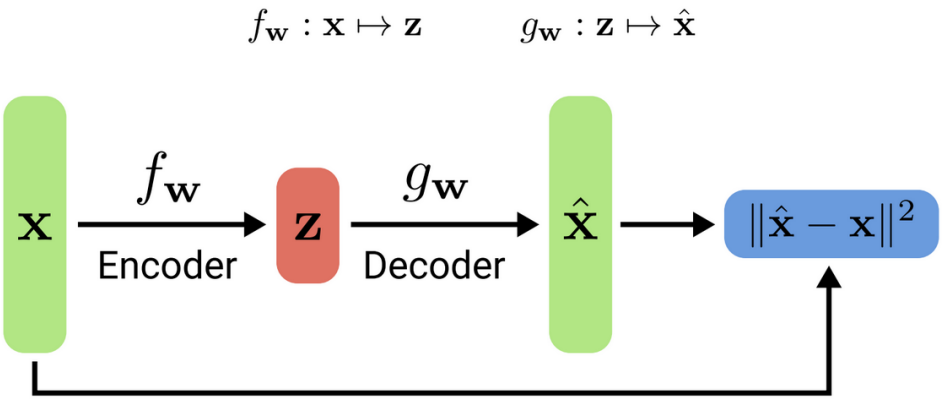

Figure 2: Visualization of an autoencoder with decoder, latent state and decoder [2, p. 7 L. 11]

This model architecture can be used for compressing data, as the latent state is typically of a lower dimensionality, and to generate new data based on sampled instances in the latent space.

(b) Briefly explain the purpose of the KL divergence.

Answer:

The Kullback-Leibler divergence KL is a distance function to measure the difference between a probability distribution P to a reference distribution Q. It measures the expected additional Shannon-information, if Q is used instead of the true distribution P. The KL-divergence is defined as follows:

$$\mathrm{KL}(P||Q) = \sum_{x \in \Omega} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{21}$$

(c) Given two discrete probability distributions Q, P. Show that $\mathrm{KL}(Q||P) = \mathrm{KL}(P||Q)$ does not hold in general.

Answer:

We show this by constructing a simple counter example based on two Bernoulli distributions.

$$P(x) = \begin{cases} 0.3 & \text{, if x = 0} \\ 0.7 & \text{, if x = 1} \end{cases} \quad Q(x) \qquad = \begin{cases} 0.5 & \text{, if x = 0} \\ 0.5 & \text{, if x = 1} \end{cases} \tag{22}$$

Now we can compute the KL-divergences in both directions:

$$\mathrm{KL}(P||Q) = 0.3 \cdot \log_2 \left(\frac{0.3}{0.5} \right) + 0.7 \cdot \log_2 \left(\frac{0.7}{0.5} \right) \tag{23}$$

$$\approx 0.1187 \tag{24}$$

$$\mathrm{KL}(Q||P) = 0.5 \cdot \log_2 \left(\frac{0.5}{0.3} \right) + 0.5 \cdot \log_2 \left(\frac{0.5}{0.7} \right) \tag{25}$$

$$\approx 0.1258 \tag{26}$$

Since $0.1187 \neq 0.1258$, we have shown that the KL-divergence is not symmetric in general.

(d) Briefly describe the re-parametrization trick and explain why it is necessary to train a VAE.

Answer:

Variational autoencoders try to minimize the negative log likelihood that is upper bounded using the ELBO. This target function separates into the variational loss and reconstruction loss. The variational loss is the KL-divergence between the distribution of the encoder for the latent state given the data and a prior on the decoder:

$$\mathrm{KL}(q(\mathbf{h}|\mathbf{x})||p(\mathbf{x})) \tag{27}$$

The deconstruction loss describes how well the decoder is able to reconstruct the input based on the distribution given by the encoder:

$$-\mathbb{E}_{h\sim q(\mathbf{h}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{h}) \tag{28}$$

To have a target for the distribution of the encoder, we use a standard Gaussian $N(\mathbf{0}, \mathbf{1})$ as variational prior. We also assume that the encoders learned distribution is a Gaussian $N(\mu, \sigma)$. This way, the latent code corresponds to just the parameters μ, σ that fully describe the distribution and the variational loss separates into a nice and simple formula:

$$\mathrm{KL}(N(\mu, \sigma)||N(\mathbf{0}, \mathbf{I}) = \frac{1}{2} e^{\sigma^2} + \mu^2 - 1 - \sigma^2 \tag{29}$$

This formula is fully differential, which makes training easy. For a forward pass, we can simply sample from $N(\mathbf{0}, \mathbf{1})$ and modify them with the current latent space parameters to generate a new latent code:

$$\mathbf{h}(\mathbf{x}) = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \cdot N(\mathbf{0}, \mathbf{1}) \tag{30}$$

Finally, the reparametrization trick is to treat the samples as constant factors, which allows to train both encoder and decoder via backpropagation. This step is important, because in the original formulation, we sample latent states inside the network directly from $N(\mu, \sigma)$, which is intractable.

(e) Name a frequently mentioned drawback of VAEs and give one example from the literature that addresses this issue.

Answer:

The main drawback of original VAEs compared to eg. GANs was that the generated images tend to be not a sharp, but slightly blurry. This was addressed by further improvements, such as the Vector-Quantized Variational Autoencoder (VQ-VAE 2) [3]. This method uses prototype vectors from a learned codebook in the latent space and samples from an autoregressive model. As a result, its generated images are much sharper and come close to the SOTA of GANs.

(f) What are the major differences of the variational autoencoders (VAEs) in comparison with standard autoencoders? Relate your answer to the two sub-losses that are optimized during VAE training.

Answer:

Autoencoders only use a reconstruction loss to minimize the difference between original inputs and decodings. VAEs, on the other hand, aim to minimize the negative log likelihood, which can be upper bounded via the ELBO into the original reconstruction loss and a variational loss that minimizes the difference between the model's distribution and a variational prior distribution.

Since this prior was chosen as $N(\mathbf{0}, \mathbf{1})$, we push the distribution of latent codes to be roughly standard Gaussian. This allows us to treat the VAEs as probabilistic generator of a data distribution and we can generate new data by simply sampling random noise from a standard Gaussian and pass them to the predicted distribution parameters to generate new latent codes.

(g) Bonus: Show that the KL-divergence between two Gaussian distributions simplifies in the following form, if the reference distribution is the standard normal:

$$\mathrm{KL}(N(\mu, \mathbf{\Sigma})||N(\mathbf{0}, \mathbf{I}) = \frac{1}{2}\sum_{i=1}^{k}\left(\sigma_i^2 + \mu_i^2 - 1 - \log(\sigma_i^2)\right) \tag{31}$$

Answer: We are given the general form of the KL-divergence:

$$\mathrm{KL}(N(\mu_1, \mathbf{\Sigma_1})||N(\mu_2, \mathbf{\Sigma_2})) = \frac{1}{2}\left\{\mathrm{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1) - k - \log\frac{\det\Sigma_2}{\det\Sigma_1}\right\} \tag{32}$$

Plugging in $N(\mu, \mathbf{\Sigma})$ and $N(\mathbf{0}, \mathbf{I}$, this yields:

$$\mathrm{KL}(N(\mu, \mathbf{\Sigma})||N(\mathbf{0}, \mathbf{I}) = \frac{1}{2}\left\{\mathrm{tr}(\mathbf{I}^{-1}\Sigma) + (-\mu)^T\mathbf{I}^{-1}(-\mu) - k + \log\frac{\det\mathbf{I}}{\det\Sigma}\right\} \tag{33}$$

$$= \frac{1}{2}\left\{\mathrm{tr}(\Sigma) + \mu^T\mu - k - \log(\det\Sigma)\right\} \tag{34}$$

Since, the covariance is a diagonal matrix $\Sigma = \mathrm{diag}(\sigma_1^2, \dots \sigma_k^2)$, the trace and log determinant simplify in the following way:

$$\mathrm{tr}(\mathbf{\Sigma}) = \sum_{i=1}^{k}\sigma_i^2 \tag{35}$$

$$\mu^T\mu = \sum_{i=1}^{k}\mu_i^2 \tag{36}$$

$$\det\Sigma = \prod_{i=1}^{k}\sigma_i^2 \tag{37}$$

$$\Rightarrow \log(\det\Sigma) = \log\prod_{i=1}^{k}\sigma_i^2 \tag{38}$$

$$= \sum_{i=1}^{k}\log\sigma_i^2 \tag{39}$$

By plugging them into the previous formula, we get:

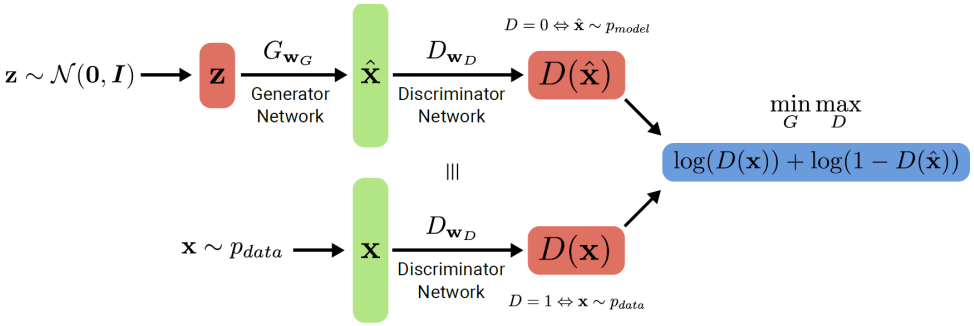

Figure 3: A Generative Adversarial Network (GAN) with generator G and discriminator D [2, pp. 10, L. 12].

$$\mathrm{KL}(N(\mu, \Sigma)||N(\mathbf{0}, \mathbf{I}) = \frac{1}{2}\left\{\mathrm{tr}(\Sigma) + \mu^T\mu - k - \log(\det\Sigma)\right\} \tag{40}$$

$$= \frac{1}{2}\left\{\sum_{i=1}^{k}\sigma_i^2 + \sum_{i=1}^{k}\mu_i^2 - \sum_{i=1}^{k}1 - \sum_{i=1}^{k}\log\sigma_i^2\right\} \tag{41}$$

$$= \frac{1}{2}\left\{\sum_{i=1}^{k}\left(\sigma_i^2 + \mu_i^2 - 1 - \log\sigma_i^2\right)\right\} \tag{42}$$

Task 03

(a) Illustrate the major structure of GANs and name the essential components. Briefly describe the principle.

Answer:

Just like VAEs, GANs are based on two major components: the generator and the discriminator. The generators goal is to generate data that is as similar as possible to the original training data. The discriminator has to distinguish between real data from the training data set and artificial data generated by the generator. The general structure of a GAN can be seen in figure 3.

If everything works out perfectly, both networks are about equally capable and improve each other over time: to fool the discriminator, the generator learns to generate more realistic artificial data. Since the generated data is getting more realistic, the discriminator has to become better and recognizing fake data. In practice, this is not as easy and GANs tend to be very hard to train, as described in the next task.

(b) Elaborate on why the training of GANs is considered as particularly difficult/challenging and how it is addressed. Comment on what happens if the discriminator dominates the generator and vice versa.

Answer:

The training process of GANs involves the joint optimization of two different neural networks two-player minimax game. The training process only works efficiently, if both networks are about equally good, because the respective training objectives should be difficult, to enforce learning, but not impossible.

If the discriminator dominates the generator, the generator cannot learn how to generate data that fools the discriminator, because the discriminator is to good at distinguishing. If the generator dominates the discriminator, the latter is not able to discriminate between real training and generated data. As a result, the training objective for the generator is far too easy and it has no incentive to generate more realistic data.

These two problems are addressed in several ways: first of all, there is an alternating optimization scheme that optimizes the discriminator for a fixed sets of steps k and afterwards the generator for a single step. This ensures the discriminator D is always close to its optimal solution, because the generator G does not change too quickly.

Especially at the beginning, when the generated data is very unrealistic, the discriminator tends to reject artificial samples with high confidence. As a result, the generator objective $\log(1 - D(G(\mathbf{z}))$ saturates, its gradient is very small and the training very slow. This is addressed with the gradient trick by replacing $\log(1 - D(G(\mathbf{z}))$ with $-\log(D(G(\mathbf{z}))$ in the beginning of the training. This function has the same target and direction of the gradient, but the gradient is much larger for small inputs. Later, we can switch back to the original target function, when its gradient becomes better.

References

[1] Mikhail Belkin et al. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[2] Andreas Geiger. *Lecture: Deep Learning*. https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/deep-learning/. 2020.

[3] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. “Generating Diverse High-Fidelity Images with VQ-VAE-2”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.