In this final exercise sheet we look back to several different topics of the lecture. You don't need python to solve the exercises. Please hand in a LaTeX-generated PDF or a jupyter-notebook with markdown cells. Don't forget to write down interim solutions.

1. **Case Study: Model design**
   You are given time series data of EEG measurements at 250 Hz, along with a vector of time stamp indices at which the proband blinked (one entry for each blink). You're now tasked to design a deep learning pipeline to detect blinks.
   (a) How would you design the pipeline? I.e., think about data splitting and encoding, type of model, loss, metric, optimizer and explain your decisions. Try to be concise and list them in the order you would work on them! (4)
   (b) What other components might be useful in the final model? (2)
   (c) For course improvements, we would like your **feedback about *this* question**. At least, tell us how much time **(hours)** you invested, if you had major problems and if you think it's useful.

   Points for Question 1: 6

2. **Network equivalence**
   Consider a two-layer dense network in which the hidden-unit nonlinear activation functions are given by logistic sigmoid functions of the form

$$\sigma(z) = (1 + \exp(-z))^{-1}.$$

   Output activation functions are identity functions.
   (a) Show that there exists an equivalent network, which computes exactly the same function but with hidden unit activation functions given by $\tanh(z)$ where the tanh function is defined as (3)

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

   Hint: First find the relation between $\sigma(z)$ and $\tanh(z)$, and then show that the parameters of the two networks differ by linear transformations.
   (b) For course improvements, we would like your **feedback about *this* question**. At least, tell us how much time **(hours)** you invested, if you had major problems and if you think it's useful.

   Points for Question 2: 3

3. **Common mistakes**
   Debug the following code snippets. They contain some common errors.
   (a) After training a few steps, `loss` becomes `Nan`. What could be two of the reasons? (2)

```
class MyModel(nn.Module):

    def __init__(self):
        super().__init__()
        self.bias = nn.Parameter(torch.randn(20))
        self.linear = nn.Linear(20, 10)

    def forward(self, x):
        return self.linear(x) + torch.log(self.bias)

model = MyModel()
loss = nn.MSELoss()(targets, model(x))
```

(b) This implementation of a normalization layer (without batch dimension) might cause a problem. What is the problem and how would you fix it? (1)

```
class ZeroMeanUnitVariance(nn.Module):

    def forward(x):
        return (x - x.mean()) / x.var().sqrt()
```

(c) For course improvements, we would like your **feedback about *this* question**. At least, tell us how much time **(hours)** you invested, if you had major problems and if you think it's useful.

Points for Question 3: 3

4. **Weight decay**

Weight decay is used for several similar regularization techniques. All of them force smaller weight values but differ in how they reduce the weights.

L^2-regularization , also called *ridge regression* or *Tikhonov regularization*, extends the loss $J(\cdot)$ by the regularization term $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$. The combined loss is used to calculate the gradient in the SGD.

$$\begin{aligned}
\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) &\leftarrow J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \lambda\Omega(\boldsymbol{\theta}) \\
\boldsymbol{g} &\leftarrow \nabla\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \\
\boldsymbol{v} &\leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v}
\end{aligned}$$

"Decoupled" weight decay reduces the parameters in update step.

That's how SGD with momentum and decoupled weight decay works.

$$\begin{aligned}
\boldsymbol{g} &\leftarrow \nabla J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \\
\boldsymbol{v} &\leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g} \\
\boldsymbol{\theta} &\leftarrow (1-\lambda)\boldsymbol{\theta} + \boldsymbol{v}
\end{aligned}$$

"Coupled" weight decay reduces the parameters in the running momentum.

$$\begin{aligned}
\boldsymbol{g} &\leftarrow \nabla J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \\
\boldsymbol{v} &\leftarrow \alpha\boldsymbol{v} - \epsilon(\boldsymbol{g} - \lambda\boldsymbol{\theta}) \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v}
\end{aligned}$$

(a) Show for *SGD with momentum* that the weight decay techniques above are equivalent. (3)

(b) Explain why this equivalence doesn't hold for adaptive gradient optimizers like Adam. (2)

(c) For course improvements, we would like your **feedback about *this* question**. At least, tell us how much time **(hours)** you invested, if you had major problems and if you think it's useful.

Points for Question 4: 5

5. **Bayesian Linear Regression**

$\mathbf{X}$ is given data with dimension $D \times N$ where D is the number of features and N is the number of data points. We will perform Bayesian Linear Regression on 2-D input data (with the 2nd dimension just for the bias) and 1-D output data.

Here

$$\mathbf{X} = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

and

$$\mathbf{y} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

Assume the prior distribution on the weights to be $p(\mathbf{w}) \sim \mathcal{N}(0, \mathbf{\Sigma}_p)$ with

$$\mathbf{\Sigma}_p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

(a) Calculate the posterior weight distribution using Bayesian Linear Regression. The formula would be: (2)

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \sim \mathcal{N}\left(\mathbf{w} = \frac{1}{\sigma_n^2}\mathbf{A}^{-1}\mathbf{X}\mathbf{y}, \mathbf{A}^{-1}\right)$$

where $\mathbf{A} = \sigma_n^{-2}\mathbf{X}\mathbf{X}^T + \mathbf{\Sigma}_p^{-1}$
and $\sigma_n^2 = 1$ is the variance of the noise in the data

(from slide 26 lecture 10 and GPML book)

(b) Plot the line corresponding to the MAP estimate for $\mathbf{w}$. (1)

(c) For course improvements, we would like your **feedback about *this* question**. At least, tell us how much time **(hours)** you invested, if you had major problems and if you think it's useful.

Points for Question 5: 3

6. **Convolution**

(a) Convolve the following input (2)

4	8	5	6	8
1	2	2	8	6
4	4	9	2	7
2	3	7	8	4
8	0	0	7	6

with the following filter (no padding and a stride of 1)

1	0
0	-1

(b) Now instead of convolution perform cross-correlation. (1)

(c) Take the result of the cross-correlation and pass it through a ReLU layer and then sum over all the elements to get the output. Now consider the target to be 1. Assuming an L1 loss between the output and the target calculate the Jacobian and Hessian of the loss with respect to the parameters of the filter (Flatten the filter in row-major order so that it's a vector). **Note:** For non-differentiable points (ReLU and L1-Loss) assume derivative is 1 when the function output is 0. (2)

(d) For course improvements, we would like your **feedback about *this* question**. At least, tell us how much time **(hours)** you invested, if you had major problems and if you think it's useful.

Points for Question 6: 5

You can achieve a total of **25 points** for this exercise. Additionally, you can achieve **1 bonus point** for answering the feedback questions.

Please send the **solution *pdf or notebook* of your group of three** via ILIAS until **28.01.2019 18:00**.

Note: Jupyter notebooks will be executed **from top to bottom**. To **avoid point deduction** check your notebook by the following steps: 1. Use the python 3 kernel (`Kernel > Change kernel > Python 3`), 2. Run the full notebook (`Kernel > Restart & Run All`), 3. Save (`File > Save and Checkpoint`).