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.

Solution:

(a)
1. **Split the data: (0.5P)** Create a training, validation and test set and lock away the test set.
2. **Metric (0.5P)** As metric to evaluate our approach, we could consider accuracy. However, there are likely significantly less samples for blinks, so the class distribution is imbalanced. Therefore, the F-score, which takes Precision and Recall into account, is more appropriate.
3. **Data encoding: (1P)** We can formulate this as a 2-class classification problem. The ratio of blink to non-blink time stamps is likely very low. Also, a blink lasts usually from 100ms to 150ms while the label only spans $\frac{1}{250} = 4$ms. To encode the data we could therefore use a one-hot encoding, which we then smooth by setting the label for data points close to points marked as blink to be larger than 0, e.g. with a Gaussian distribution. Also, always visualize the data during all steps! And if you have domain experts at hand, talk to them!
4. **Model:(1P)** We are dealing with time series data, which suggests using an LSTM or a convolutional model. As individual blinks are unrelated to each other and the blinks durations are probably very similar, a convolutional model consisting of multiple 1D convolution layers fits well. As a blink spans over a wide window of measurement points, we could experiment with stride > 1 or dilated convolutions.
5. **Loss:(0.5P)** We formulated the problem as a classification problem, so using the cross entropy loss as loss function is a good choice.
6. **Optimizer:(0.5P)** Stochastic gradient descent with Momentum (SGD) or ADAM is usually a good choice. If the model is really big, SGD might be the better choice, as it stores less moving averages and is therefore less memory demanding.

(b) **Additional components:** Search for hyperparameters (here we need the validation set). If the initial model is overfitting, we can consider regularization, e.g. weight decay, $L2/L1$-regularization and dropout. We can try to find more EEG data or augment the data. We can consider ensembling. If the model should run on less capable hardware, we can consider quantization (rescaling the weights from 32 bit to 8 bit representations).

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.

Solution:

(a) The relation between the activation functions is

$$\begin{aligned}
\tanh(z) &= \frac{(e^z - e^{-z}) - \overbrace{(e^z - e^z)}^{0}}{e^z + e^{-z}} \\
&= \frac{2e^z}{e^z + e^{-z}} - 1 \\
&= \frac{2e^z \cdot 1}{e^z \cdot (1 + e^{-2z})} - 1 \\
&= 2\sigma(2z) - 1.
\end{aligned}$$

The described MLP with weight vectors $w^{(1)}, w^{(2)}$ and activation functions $g(\cdot)$ can be formulated as

$$\begin{aligned}
z_j &= \sum_{i>0} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \\
y_k &= \sum_{j>0} w_{kj}^{(2)} g(z_j) + w_{k0}^{(2)}.
\end{aligned}$$

If the two networks are equivalent, we come up with

$$\begin{aligned}
\sum_{j>0} w_{kj}^{(2,\sigma)} \sigma(z_j^{(\sigma)}) + w_{k0}^{(2,\sigma)} &= \sum_{j>0} w_{kj}^{(2,\tanh)} \tanh(z_j^{(\tanh)}) + w_{k0}^{(2,\tanh)} \\
&= \sum_{j>0} w_{kj}^{(2,\tanh)} (2\sigma(2z_j^{(\tanh)}) - 1) + w_{k0}^{(2,\tanh)} \\
&= \sum_{j>0} w_{kj}^{(2,\tanh)} 2\sigma(2z_j^{(\tanh)}) - \sum_{j>0} w_{kj}^{(2,\tanh)} + w_{k0}^{(2,\tanh)}.
\end{aligned}$$

This results in the following three constraints ($j > 0$):

$$w_{kj}^{(2,\sigma)} = 2w_{kj}^{(2,\tanh)} \tag{1}$$
$$w_{k0}^{(2,\sigma)} = -\sum_{j>0} w_{kj}^{(2,\tanh)} + w_{k0}^{(2,\tanh)} \tag{2}$$
$$z_{kj}^{(\sigma)} = 2z_{kj}^{(\tanh)} \tag{3}$$

The constraints 1 and 2 can directly be satisfied by linear transformation of the weights. To resolve constaint3, we can insert the first layer

$$\begin{aligned} & z_{kj}^{(\sigma)} = 2z_{kj}^{(\tanh)} \\ \Rightarrow & \sum_{i>0} w_{ji}^{(1,\sigma)} x_i + w_{j0}^{(1,\sigma)} = \sum_{i>0} 2w_{ji}^{(1,\tanh)} x_i + 2w_{j0}^{(1,\tanh)}. \end{aligned}$$

The resulting constraint 4 (with $i \geq 0$) is also trivial to satisfy.

$$w_{ji}^{(1,\sigma)} = 2w_{ji}^{(1,\tanh)} \tag{4}$$

Acknowledgement: This exercise is based on Exercise 5.1 of the PRML book (Bishop, 2016).

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.

Solution:

(a) One problem is, that we take the logarithm of the trainable parameter `self.bias`. If the parameter gets negative, the log will be NaN. However, another reason could be that your input tensor or target tensor is corrupted and contains NaN values!

(b) If the variance of x is zero, this becomes infinity. We can fix this by adding a small epsilon, i.e., `eps=1e-6; return (x - x.mean()) / (x.var() + eps).sqrt()`

Points for Question 3: 3

4. **Weight decay**

Weight decay is used for several similar regularization techniques. All of them enforce smaller weight values but differ in how they reduce the weights. In the following, λ denotes the weight decay factor.

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. Can you use the same decay factor λ? (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.

Solution:

(a) They are equivalent for $\lambda_a = \frac{\lambda_b}{\epsilon} = -\lambda_c$. Be aware that this might play a role when you re-implent a paper and you just copy the hyperparameters!
Proof:

L^2-regularization

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

Insert $\boldsymbol{v}$ and $\boldsymbol{g}$.

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

Decoupled weight decay Insert $\boldsymbol{v}$ and $\boldsymbol{g}$.

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

"Coupled" weight decay Insert $\boldsymbol{v}$ and $\boldsymbol{g}$.

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

(b) Adaptive gradient algorithms rescale the gradient components with individual values $\boldsymbol{M}$. For L^2-regularization the weight decay part is also rescaled, because it is part of the loss-gradient. The equivalence would only hold for $\boldsymbol{M} = k\boldsymbol{I}$. This is very unlikely because the entries of $\boldsymbol{M}$ are adapted independently at each optimization step.

L^2-regularization

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{M} \nabla \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) + \dots \\ \Rightarrow \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{M} \nabla J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) - \epsilon \boldsymbol{M} \lambda_a \boldsymbol{\theta} + \dots \end{aligned}$$

Decoupled weight decay

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{M} \nabla J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) - \epsilon \lambda_b \boldsymbol{\theta} + \dots$$

Points for Question 4: 5

5. **Bayesian Linear Regression**

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.

Solution:

(a) By plugging the values in to the above formulae we get,

$$\mathbf{A} = \begin{bmatrix} 14 & 6 \\ 6 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ 6 & 4 \end{bmatrix}$$

and

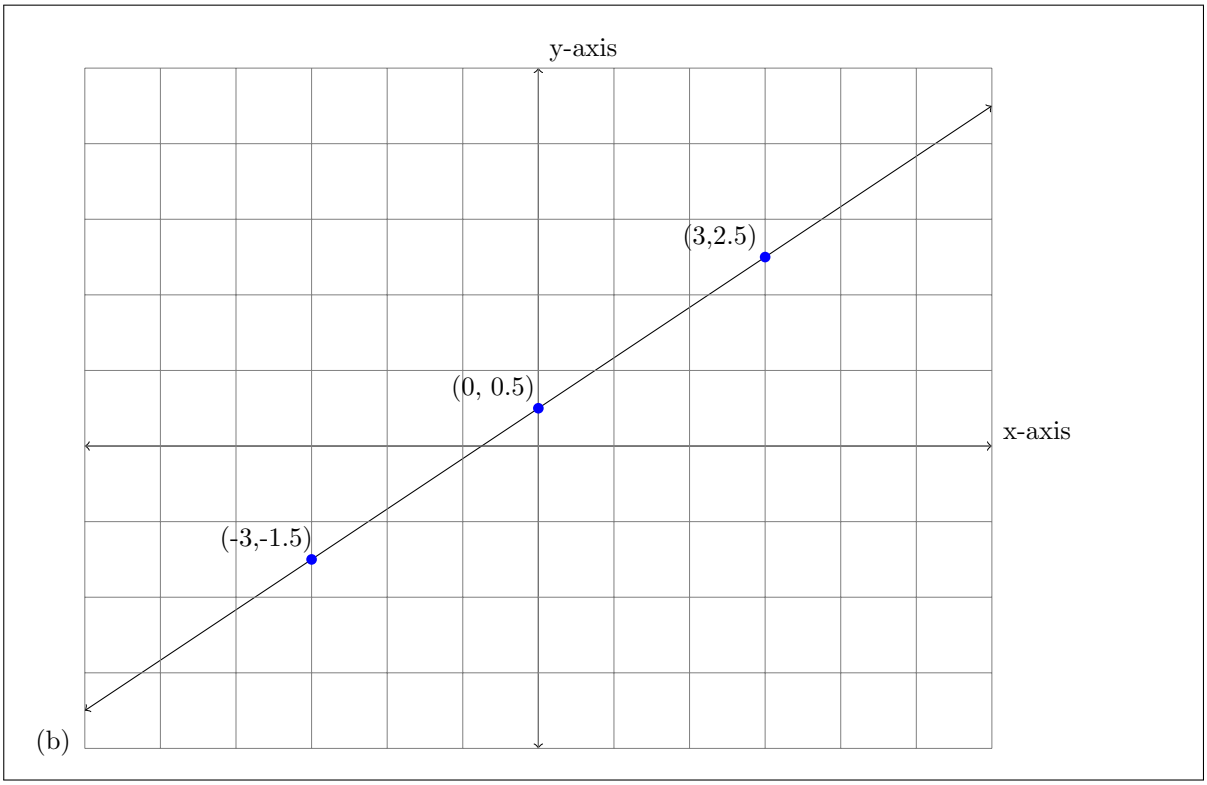
$$\mathbf{A}^{-1} = \frac{1}{24}\begin{bmatrix} 4 & -6 \\ -6 & 15 \end{bmatrix}$$

and

$$\mathbf{X}\mathbf{y} = \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

which implies

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \sim \mathcal{N}\left(\mathbf{w} = \frac{1}{24}\begin{bmatrix} 16 \\ 12 \end{bmatrix}, \frac{1}{24}\begin{bmatrix} 4 & -6 \\ -6 & 15 \end{bmatrix}\right)$$



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.

Solution:

(a) Let the input be

$$\mathbf{X} \text{ with } i, j^{th} \text{ element } x_{i,j}$$

Let the filter be

$w_{0,0}$	$w_{0,1}$
$w_{1,0}$	$w_{1,1}$

and the output of the cross-correlation be

$$\mathbf{Z} \text{ with } i, j^{th} \text{ element } z_{i,j}$$
$$\text{where } z_{i,j} = w_{0,0} * x_{i,j} + w_{0,1} * x_{i,j+1} + w_{1,0} * x_{i+1,j} + w_{1,1} * x_{i+1,j+1}$$

(Convolution will be the same except the filter will be flipped 180°.)

and the result after the ReLU activation be

$$\mathbf{A} = \text{ReLU}(\mathbf{Z}) \text{ with } i, j^{th} \text{ element } a_{i,j}$$

Then the output $\hat{y}$ of this trivial network will be

$$\hat{y} = \sum_{i,j} a_{i,j}$$

and loss

$$L = |\hat{y} - y|$$

The convolution will here be equal to

-2	-6	3	0
3	7	0	-1
-1	3	-1	2
-2	-3	0	-2

(b) The cross-correlation will be the negative of the convolution since the filter flipped by 180°is the negative of the original filter

2	6	-3	0
-3	-7	0	1
1	-3	1	-2
2	3	0	2

(c) Backpropagating the errors we get,

$$\frac{\delta L}{\delta \hat{y}} = \begin{cases} 1, & \text{if } y \leq \hat{y} \\ -1, & \text{otherwise} \end{cases}$$

$$\frac{\delta \hat{y}}{\delta a_{i,j}} = 1$$

and

$$\frac{\delta a_{i,j}}{\delta z_{i,j}} = \begin{cases} 1, & \text{if } z_{i,j} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

and

$$\frac{\delta z_{i,j}}{\delta w_{k,l}} = x_{i,j} \text{where } i, j, k \text{ and } l \text{ are constrained by the filter size and input size.}$$

By the chain rule and just before summing all the contributions to the derivatives of the loss for each parameter, we get

$$\frac{\delta \mathbf{L}}{\delta z_{i,j}} \frac{\delta z_{i,j}}{\delta w_{0,0}} =$$

4	8	0	0
0	0	0	8
4	0	9	0
2	3	0	8

$$\frac{\delta \mathbf{L}}{\delta z_{i,j}} \frac{\delta z_{i,j}}{\delta w_{0,1}} =$$

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

$$\frac{\delta \mathbf{L}}{\delta z_{i,j}} \frac{\delta z_{i,j}}{\delta w_{1,0}} =$$

1	2	0	0
0	0	0	2
2	0	7	0
8	0	0	7

$$\frac{\delta \mathbf{L}}{\delta z_{i,j}} \frac{\delta z_{i,j}}{\delta w_{1,1}} =$$

2	2	0	0
0	0	0	7
3	0	8	0
0	0	0	6

and finally summing up the contributions, we get the Jacobian as

46
39
29
28

The Hessian will be **0** because all the terms are lower than 2nd order in each of the parameters.

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`).