altklausur

1. Exercise

Multiple Choice questions about all topics (properties of ReLU, logcosh, how many parameters does torch.nn.maxPool2d(3) add to a NN, main problem of perceptron loss, properties of ResNets, U-nets, pooling layers)

2. Exercise

Given $f(x_1, x_2)$, $h = \max(0, x_1)$, $y = \max(x_2, h)$
Draw the 4 linear pieces, given the first one $((x_1 > 0) \wedge (x_2 > h))$ as an example.
Draw the decision boundary for all 4 linear pieces, given the first one as an example.

3. Exercise (Optimization)

Very similar to optimization exercise sheet. Hessian, condition number, advantages and disadvantages of adding noise to data to increase variance.

4. Exercise (RNNs)

Draw RNN, compute $\delta L/\delta y$, state $\delta L/\delta \theta$, compute $\delta L/\delta \theta$ given $\delta 1, \delta 2, \delta 3$ just like in RNN exercise sheet. Also just write g' for the derivative of g.

$$h1 = g(x1, h0, \theta)$$

$$h2 = g(x2, h1, \theta)$$

$$h3 = g(x3, h2, \theta)$$

$$y = h3$$

$$L(y, t) = max(0, -yt)$$

5. Exercise (Autoencoder programming)

1. Given a dataset of 500 rgb images, train a NN that colors greyscale images. Dataset X is Nx3xSxS, where N is the number of images and S is the size of an image. Transform the rgb images to greyscale images by taking the average of the rgb values as greyscale. Write the training loop, given the skeleton code below.

```
def train(X, model):
optim = torch.nn.SGD(model.parameters(), lr=0.004)
# Space for code
loss_fn = torch.nn.MSELoss()
X_grey = X.mean(dim=1, keepdim=True)

for i in range(1000):
    optim.zero_grad()
    # Space for code
    output = model(X_grey)
    loss = loss_fn(output, X)
    loss.backward()

    optim.step()
```

2. Now we want to change the task from coloring images to filling images where there is missing data. Instead of turning the images into greyscale, simply zero out a 16x16 area in the middle of the image. Indicate which lines to change from the code you've written above and give the new code.

```
def train(X, model):
optim = torch.nn.SGD(model.parameters(), lr=0.004)
# Space for code
loss_fn = torch.nn.MSELoss()
X_missing = X.clone()
S = X.shape[2]
start = S//2 - 8
X_missing[:, :, start:start+16, start:start+16]
```

```
for i in range(1000):
    optim.zero_grad()
    # Space for code
     output = model(X-missing)
     loss = loss_fn(output, X)
     loss.backward

    optim.step()
```

3. Explain the limitations and problems that can arise with this appraoch and how to fix them (1-2 sentences).

ungefähr so, keine ahnung ob das mit dem `X.mean()` so passt aber das sollte genug Raum zum anfechten bieten, falls sie da Punkte abziehen.

```
def train(X, model):
optim = torch.nn.SGD(model.parameters(), lr=0.004)
loss_fn = torch.nn.MSELoss()
X_gray = X.mean(dim=1, keepdim=True)
for i in range(1000):
    optim.zero_grad()
    out = model(X_gray)
    loss = loss_fn(out, X)
    loss.backward()
    optim.step()
```

Question 1:
Multiple Choice fragen, alle Themen können drankommen

Question 2:
Linear boundary mahlen für eine Funktion

Question 3:
Derivative of an error function + upper bound it

Question 4: RNN mahlen und die Derivative und multi-chain rule davon berechnen

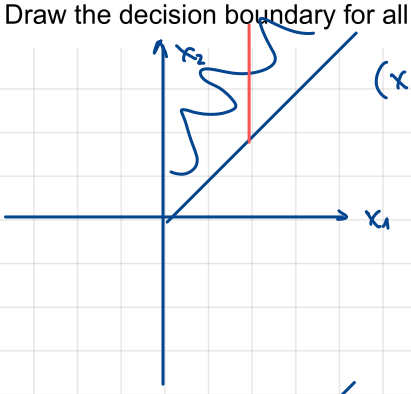
Question 5: Autoencoder programming

2. Exercise

Given $f(x_1, x_2)$, $h = \max(0, x_1)$, $y = \max(x_2, h)$

Draw the 4 linear pieces, given the first one $((x_1 > 0) \wedge (x_2 > h))$ as an example.

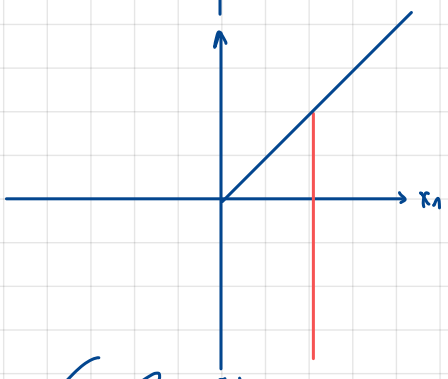
Draw the decision boundary for all 4 linear pieces, given the first one as an example.



$(x_1 > 0) \wedge (x_2 > h)$

$h = x_1$

$x_2 > x_1$



$(x_1 > 0) \wedge (x_2 < h)$

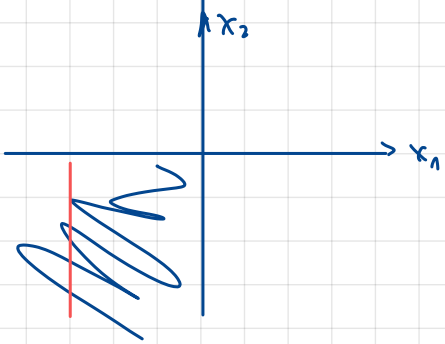
$h = x_1$

$x_2 < x_1$

$(x_1 < 0) \wedge (x_2 > h)$

$h = 0$

$x_2 > 0$



$(x_1 < 0) \wedge (x_2 < h)$

$h = 0$

$x_2 < 0$

$y = x_2$