Machine Learning 2020 Exam

Note: the points are marked individually as 🍊 in the answers, but this just gives a general indication where we provided the points during grading.

Problem 1: A bit of code

Problem 1a:

(a) ❹ The following network is given:

```
Net=Sequential()
Net.add(Conv1D(filters=1,kernel_size=3,padding="same",activation="relu"))
```

Which of the following mappings can be (in principle) successfully learned by this neural network? Briefly explain your answer in each case! [padding "same" simply guarantees that the output is the same length as the input, by adding zeros at the boundaries of the input as needed before applying the convolution]

A [0,0,0,1,0,0] $\mapsto$ [0,0,0.2,1.4,0.3,0]

B [0,0,5,0,0,0] $\mapsto$ [0.1,0.2,0.5,0.3,0.1,0]

C [1,0,0,0,0,0] $\mapsto$ [0.3,-0.1,0,0,0,0]

D [0,1,0,-1,0,0] $\mapsto$ [0.1,0,0,1.8,0,0]

Answer 1a:

Note: kernel_size=3 means any result entry can be influenced by the input slots 1 to the left and 1 to the right. relu means the output must be non-negative.

A: 🍊 **yes**, only involves elements within ±1 slots and result is >=0
B: 🍊 **no**, would need larger kernel_size
C: 🍊 **no**, output is sometimes negative, which is forbidden by relu
D: 🍊 **yes**, the kernel [0,-1.8,0.1] (and bias zero) would do the job (e.g. in the first slot: applying the kernel yields 0.1 and applying relu to that also gives 0.1; second slot: kernel gives -1.8 which relu turns to zero, etc.)

Problem 1b:

(b) ❽ You want to train a neural network that is able to locate the center z of a Lorentzian function of arbitrary amplitude, location, and width, $f(x) = \frac{A}{[(x-z)/\sigma]^2+1}$, where f is provided as a 1D array input (of length N) and x is in the interval [-3,3]. Write a python program (or alternatively a program in any other language) that would generate a training batch y_input, y_target of given batchsize that could be passed to Net.train_on_batch(y_input,y_target). When using python, you might want to use any of the commands

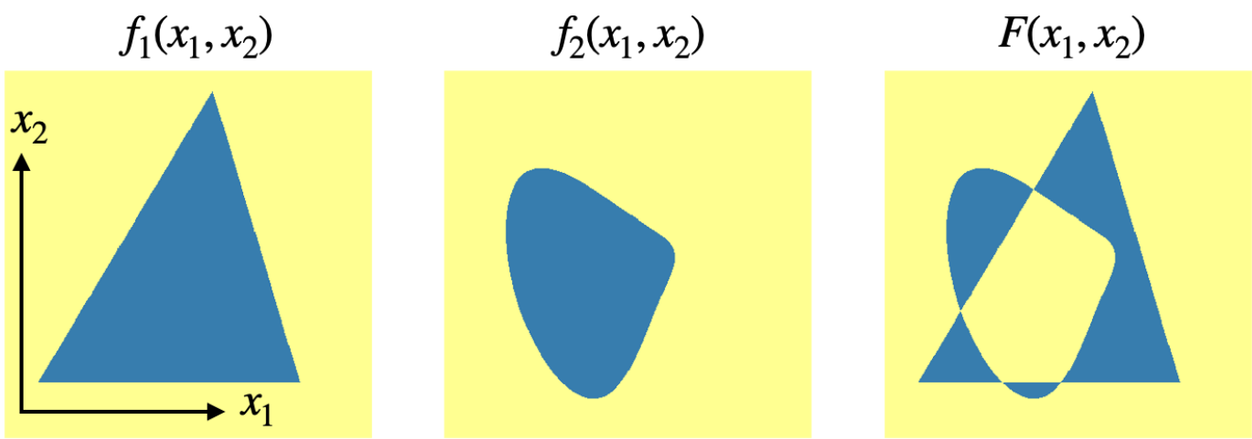
```
np.zeros([<dim1>,<dim2>,...]),

np.linspace(x_min,x_max,N),

np.random.uniform(low=value_min,high=value_max,size=array_length),
```

as well as the array dimension extension operation (for example B[None,:] which nominally adds a first dimension of arbitrary length to the vector B). You can choose and name the parameters arbitrarily, as long as they are not given here. Comment your program!

Answer 1b:

```
N=100 # number of pixels in the x grid
batchsize=5 # batch size
x=np.linspace(-3,3,N) 🍊# the x grid
A=np.random.uniform(-5.,5.,size=batchsize) 🍊# amplitudes
sigma=np.random.uniform(0.1,6.,size=batchsize) 🍊# widths
z=np.random.uniform(-3.,3.,size=batchsize) 🍊# center positions
# the following calculates y_input and
# in doing so blows up these parameter arrays in the
# second dimension, which will be used to represent the
# x-coordinates on which the functions are defined:
# 🍊 for correct expression overall, 🍊 for correct use of the array-extension ope
y_input=A[:,None]/(((x[None,:]-z[:,None])/sigma[:,None])**2+1)
# the target array: only a single number for each batch sample:
y_target=np.zeros([batchsize,1]) 🍊
# this number is set to the sigma parameter, which the
# network should be trained to extract:
y_target[:,0]=z 🍊
```

Problem 2: Constructing and analyzing neural networks



Problem 2a:

(a) ❻ Construct a neural network that, when given the values f_1 and f_2 as its two inputs (see picture: dark = 1, bright = 0), would generate as output the function F. The nonlinearity should be the step function, i.e. $\theta(y) = 1$ for $y > 0$ and $\theta(y) = 0$ for $y \leq 0$. Write down the full expressions that calculate the individual neuron values from the neuron values in the layer below (introducing some convenient notation for those neuron values) and sketch the network layout.

Answer 2a:

Given f_1 and f_2, we observe that F is the XOR function, i.e. it is 1 if and only if exactly one of the two inputs f_1 and f_2 is 1.

One possible way to construct an XOR network for inputs a and b is to check for "**(a AND (NOT b)) OR (b and (NOT a))**". Both AND and OR operations can be produced using the step function and a suitable bias.

Mathematical expression (superscript indicates layer, where 1 is the first hidden layer, i.e. the layer after the input):

🍊 for the right principle (e.g. explanation of XOR, even if rest is wrong)

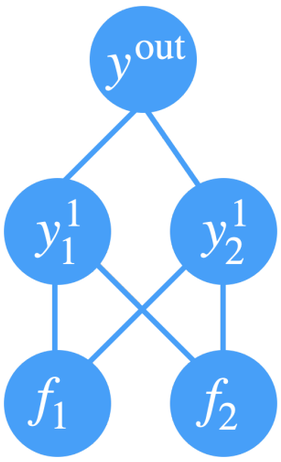
$y_1^1 = \theta(f_1 - f_2 - 0.5)$ 🍊
(will be 1 iff $f_1 = 1, f_2 = 0$)

$y_2^1 = \theta(f_2 - f_1 - 0.5)$ 🍊
(the opposite)

and then the output neuron:

$F = y^{\text{out}} = \theta(y_1^1 + y_2^1 - 0.5)$ 🍊
(will be 1 if one or the other condition above is fulfilled, but not if both are violated, i.e if both results above are zero)

Sketch: 🍊🍊



Problem 2b:

(b) ❸ Consider a convolutional network in 1D, where $y_j^{\text{out}} = \theta(\sum_{k=-1}^{+1} w_k y_{j+k}^{\text{in}} + b)$. Suggest one possible choice for the kernel weights w_k and the bias b in order to produce the following mappings:
[0 0 0 1 0 0] to [0 0 0 0 0 0],
[0 0 0 -0.5 0 0] to [0 0 0 0 0 0],
[0 0 0 -2 0 0] to [0 0 0 1 0 0],
[0 0 1 0 1 0] to [0 0 0 1 0 0].

Explain your choice! (the solution is not unique, but certainly constrained)

Answer 2b:

The third and fourth result show that the kernel cannot be zero. The fourth shows that it must contain elements away from the diagonal, i.e. where $k \neq 0$. The third shows it must have something on the diagonal, but it must be negative. The first and second together show that there must also be a bias (because otherwise either one or the other should give a nonzero outcome).

🍊 for a reasonable explanation

Possible ansatz:

(note: many possible solutions, see below for necessary conditions!)

🍊🍊 (one point even if not everything is completely correct but at least some of the values make sense, i.e. only one condition below violated by one choice)

$w_{k=0} = -1, b = -1.5$
$w_{k=\pm 1} = 1$

Check:

```
fourth slot of output:

1*(-1)-1.5 < 0 => 0
(-0.5*(-1))-1.5 <0 => 0
(-2)*(-1)-1.5 >0 => 1
1*1 + 1*1 -1.5 > 0 => 1

third or fifth slots:

1*1 - 1.5 < 0 => 0
1*(-0.5) -1.5 <0 => 0
1*(-2) - 1.5<0 => 0
1*(-1) - 1.5 < 0 => 0
```

More generally, the constraints that can be derived from the given mappings are:

$$b \leq 0$$

$$w_0 \in [2b, \frac{b}{2}]$$

$$w_{-1} + w_1 + b > 0$$

Problem 3: Reinforcement Learning

A robot is faced with the task of moving towards a target. At any given time step, it gets an input signal (state) s=-1,1 depending on whether it is to the left of the target or to the right of it. Its actions are a=-1,1 for moving left and moving right. At each time step it gets a reward r=1 if it moves towards the target, and -1 otherwise.

Let us parametrize the policy as $\pi_\theta(a|s) = \sigma(a\theta_s)$ where $\theta = (\theta_{s=-1}, \theta_{s=+1})$ is a two-dimensional parameter. Here $\sigma(x) = \frac{1}{e^{-x}+1}$ is the sigmoid.

Problem 3a:

(a) ❹ Confirm that the policy probability distribution is properly normalized, for a fixed state s. Calculate the gradient $\frac{\partial}{\partial\theta_s}\ln\pi_\theta(a|s)$. Plot this for a=+1 and s=+1 versus θ_1.

Answer 3a:

Normalization at fixed s: 🍊

$\sigma(+\theta_s) + \sigma(-\theta_s) = ?$

In general:

$$\sigma(x) + \sigma(-x) = \frac{1}{e^{-x}+1} + \frac{1}{e^{x}+1} = \frac{e^x+1+e^{-x}+1}{(e^x+1)(e^{-x}+1)} = \frac{e^x+e^{-x}+2}{e^x+e^{-x}+2} = 1.$$

q.e.d.

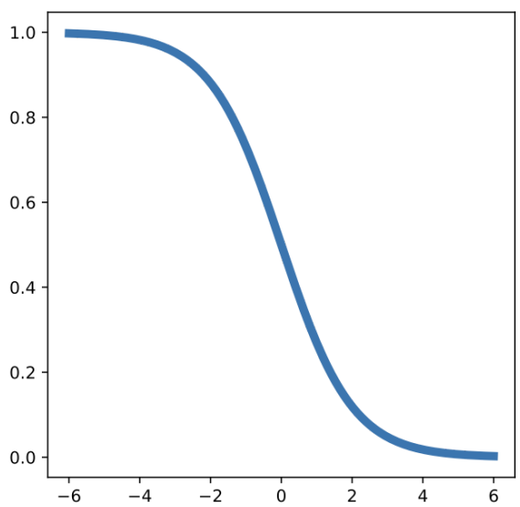
Gradient: 🍊🍊

$$\frac{\partial}{\partial\theta_s}\ln\pi_\theta(a|s) = \frac{\frac{\partial}{\partial\theta_s}\pi_\theta(a|s)}{\pi_\theta(a|s)} = \ldots = a\sigma(-a\theta_s)$$

(where we used $\partial_x\sigma(x) = \sigma(x)\sigma(-x)$)

Plot: 🍊 (right plot for any answer at previous point)

a reverted sigmoid (i.e. horizontal axis flipped)



Problem 3b:

(b) ❼ We want to apply the policy gradient update. We will do it in a simplified form, where we take an extremely discounted reward, i.e. we only care for the reward at the present time step ("greedy" approach):

$$\delta\theta_s = \eta \sum_{t=1}^{T} \left\langle r_t \frac{\partial}{\partial \theta_s} \ln \pi_\theta(a_t|s_t) \right\rangle$$

The average is over all trajectories. Assume that, statistically, half of the time the state s_t is +1, the other half of the time it is -1. Obtain the explicit equation for $\delta\theta$ as a function of θ. Sketch the flow diagram in the (θ_{-1}, θ_1) plane that shows in which direction the update will move the parameter θ. What are the fixed points? Which of them is the stable (attractive) fixed point that the reinforcement learning will tend to? Does it make sense?

Answer 3b:

For fixed s, the gradient will be nonzero only half of the time, because this state is present only half of the time (by stated assumption). And the reward is $r = -sa$, because you have to move towards the target (i.e. s and a have to have opposite sign).

Thus, multiplying the expression for the logarithmic derivative (above) with the policy probability itself and adding up both possible actions $a = \pm 1$, we get: 🍊🍊🍊

$$\delta\theta_s = \eta \frac{T}{2}(-s) 2\sigma(\theta_s)\sigma(-\theta_s) = -s\eta T \sigma(\theta_s)\sigma(-\theta_s)$$

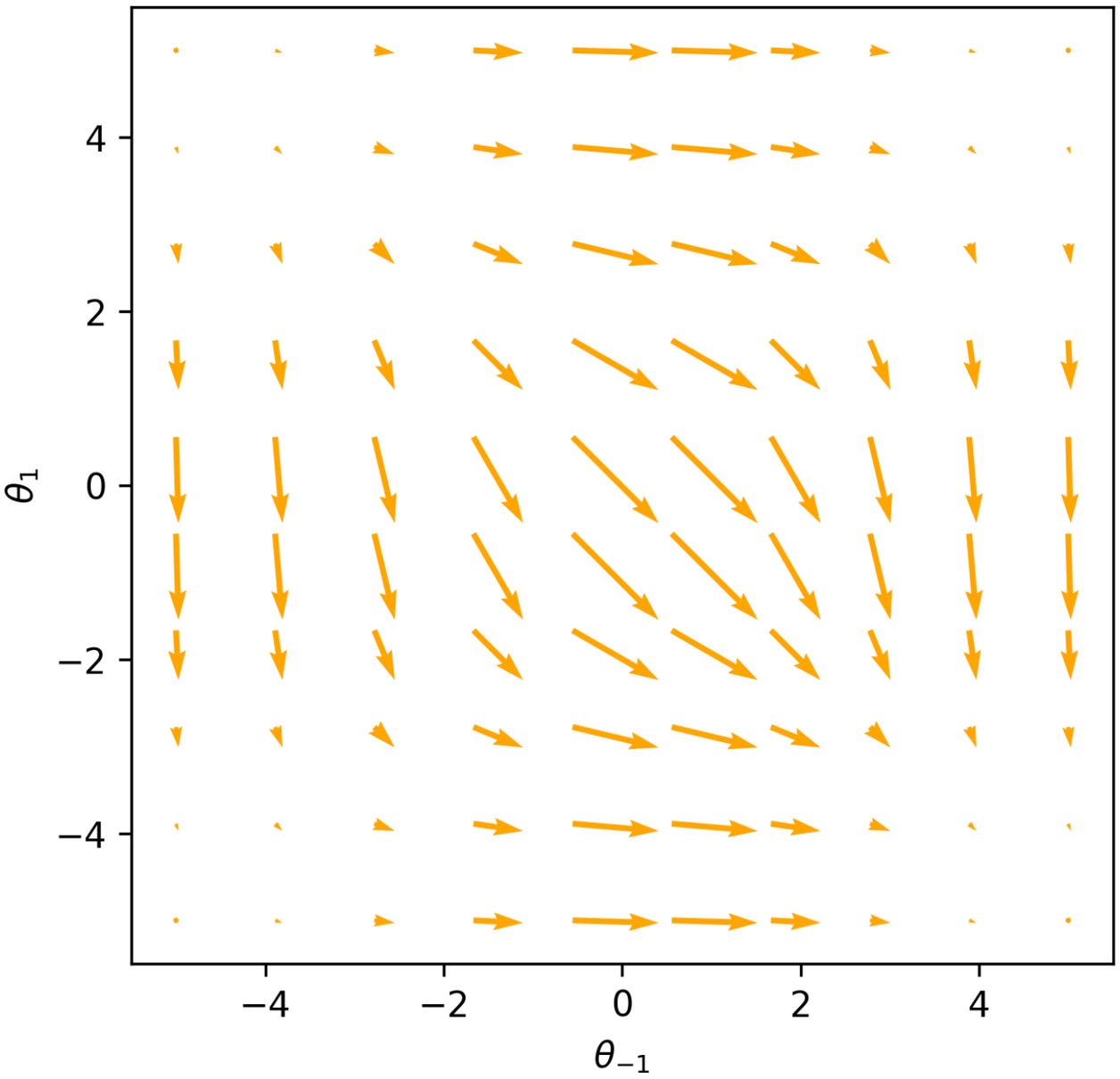
(the a^2 inside $ra = sa^2$ just becomes 1, and the '2' factor comes from adding up the contributions for both a values, each of which gives the same expression)

Note already that for $s = +1$ the θ_1 will become smaller and become negative, which is what is needed to prefer $a = -1$: that makes sense (moving towards the target)! 🍊

Thus, fixed points are $\theta_1 = -\infty, \theta_{-1} = +\infty$ (stable/attractive), but also $\theta_1 = +\infty, \theta_{-1} = -\infty$ (unstable). 🍊

Sketch: 🍊🍊

The product $\sigma(x)\sigma(-x)$ is maximal when x is around 0. So this reasoning now applies for both directions $\theta_{\pm 1}$. Note the opposite sign (as explained above). Thus the plot is:



Problem 4: Solving various problems using deep learning

You have available: An autoencoder AUTO1D that maps a 1D vector into a 1D vector, a recurrent network called RECU that maps a temporal sequence into another temporal sequence, and a convolutional network CNN2D that maps a 2D image into another 2D image.

(description of problems, see below)

Describe for each of these cases which of the networks you would use. Describe also, for each of these cases, how you would prepare the input and the output for the training samples, and how you would choose the cost function (no code is needed, but be clear in your statements).

Problem 4a:

(a) ❸ Take as input a string like “13+04=…” and produce as output “13+04=17”.

Answer 4a:

Take recurrent network **RECU**🍊, and represent each character as input during a given time-slot, as one-hot encoding. The inputs are arrays of dimension batchsize x number_of_time_steps x 13, where 13 is the number of input neurons in each time step,

because we have 13 different characters (0,1,2,…,9,+,=,.). The number_of_time_steps in this example would be 2+1+2+1+2=8. Each **input** sample is a string (converted to one-hot encoding) of the sort xx+yy=… where xx and yy are chosen as random integers🍊 (say, below 50, to obtain only two-digit results, otherwise the number_of_time_steps has to be larger, so the answer can have three digits). The correct, desired output is the same string, but with the correct result filled in. The **cost function** could be categorical cross-entropy (quadratic distance is also OK, but not quite as good)🍊.

Note: In principle, maybe the input could also be just a single neuron instead of 13, with a different value for each symbol, but this encoding will be much harder to learn.

Problem 4b:

(b) ❸ Take as input a black-and-white photograph and turn it into a colorized image.

Answer 4b:

Use a CNN2D🍊 convolutional network, with one input channel (black-and-white) and three output channels (red,green,blue). As training samples, use lots of color images, but use those images as the correct output, and their black-and-white version as input🍊. Use quadratic deviation (mean square average)🍊 as cost function.

Problem 4c:

(c) ❸ Take as input a time-trace (as obtained from physical experiments) and obtain a low-dimensional representation vector. This vector can then be used to approximately reconstruct the trace or later become the input in a classification neural network.

Answer 4c:

Take the AUTO1D🍊 autoencoder, where the time-trace is represented as a vector (each slot is a different time), and train it to reproduce the input time trace🍊 (with quadratic deviation cost function) despite a small bottleneck of a few neurons. Later, after training, take as output vector the values of the neurons in this small bottleneck layer🍊 (which are then a low-dimensional representation).

Problem 4 bonus:

❶ Moreover, for the time-traces, suppose you have a hypothesis that they can be clustered into a few different categories. Which well-known machine learning technique could you use to check this hypothesis?

Answer Problem 4 bonus:

One could use the t-SNE🍊 method where one provides a large amount of time traces as samples (where each trace becomes a ‘point’ in a high-dimensional space). t-SNE will project this down to a low-dimensional space (e.g. 2D), trying to keep nearby high-dimensional points

also close together in low dimensions. If there are really only a few clusters, one should be able to recognize them in the low-dimensional representation by visual inspection.