Last Name ..... First Name.....

Artificial Neural Networks: Exam
5th of August 2020

- Keep your bag next to your chair, but do not open it during the exam.
- Write your name in legible letters on top of this page.
- The exam lasts 180 min.
- Write **all** your answers in a legible way on the exam (no extra sheets).
- No documentation is allowed (no textbook, no slides), except **one page A5 of handwritten notes**.
- No calculator is allowed.
- Have your student card displayed in front of you on your desk.
- **Check that your exam has 17 pages; page 18 is empty; page 19 is a detachable illustration.**

Evaluation:

1. ..... / 9 pts (Section 1, Quiz-questions)
2. ..... / 9 pts (Section 2, Debugging of RL)
3. ..... / 14 pts (includes 2 bonus points, Section 3, Nesterov Momentum)
4. ..... / 12 pts (Section 4, AutoDiff)
5. ..... / 12 pts (Section 5, RL theory)

Total: ..... / 56 pts (54 pts + 2 bonus pts)

Definitions and notations

The symbol η is reserved for the learning rate.

Throughout the exam, a data base for supervised learning will be denoted by $(\mathbf{x}^\mu, \mathbf{t}^\mu)$ with $1 \leq \mu \leq P$ where $\mathbf{t}^\mu$ is the target output. The actual output of the artificial neural network will be denoted by $\hat{\mathbf{y}}$ in general and $\hat{\mathbf{y}}^\mu$ to denote the output in response to input pattern μ. Bold face symbols refer to vectors, normal face to a single component or a single input/output. Unless noted otherwise, the input is N-dimensional: $\mathbf{x}^\mu \in R^N$

The total input to a unit i is denoted by $a_i = \sum_j w_{ij} x_j - \theta_i$ with weights w_{ij} and the output of that same unit by $g(a_i)$. The function g is called the gain function. Sometimes the threshold is made explicit by a symbol θ; sometimes it is implicit in the weights.

In the context of reinforcement learning, the symbol a refers to an action; the symbols r and R to a reward; the symbol s to a discrete state; and the symbol γ to a discount rate. If the input space is continuous then inputs are also written as $\mathbf{x}$.

How to give answers

The first section contains 9 Yes-No question. For each question, you have **three possibilities: Tick yes, or no, or nothing.** Every correct answer gives one positive point, every wrong answer one negative point, and no answer no point. If the final count (with this procedure) across all questions is below zero, we give zero points. With this procedure random guesses are subtracted and if all N questions are correctly answered you receive N points.

The remaining sections involve calculations. **Please write the answers in the space provided for that purpose.**

We also provide some free space for calculations. We will not look at these parts for grading. You can also use colored paper for side calculations. We will not look at the colored paper.

1 9 Yes-No questions (9 points; explanation on the previous page)

1. A cross-entropy error function for multi-class output together with one-hot coding of the target (unit k codes for class k) and softmax function for the output guarantees that the value of the output unit k is equal to the true posterior probability $P(C_k|\mathbf{x})$, once the optimization algorithm has converged to the minimum.

   [ ] Yes or No [ ]

2. For a simple network with a single output (with monotone nonlinearity) and no hidden layer, the stochastic gradient descent update rule on an error function $E = \sum_\mu [t^\mu - \hat{y}^\mu]^4$ can be written as $\Delta\mathbf{w}(\mu) = \beta(\hat{y}^\mu)[t^\mu - \hat{y}^\mu]\mathbf{x}^\mu$, where $\beta(\hat{y}^\mu)$ is only a function of $\hat{y}^\mu$ and does not depend on t^μ.

   [ ] Yes or No [ ]

3. In a network with two hidden layers of $n = 100$ neurons each there are more than $(100!)^2$ saddle points.

   [ ] Yes or No [ ]

4. Standard 1-step SARSA is a TD algorithm, but SARSA with eligibility trace is not.

   [ ] Yes or No [ ]

5. If during one specific episode (i.e., a finite sequence of 'states, actions, rewards' chosen according to some policy) consisting of 1000 steps, a specific Q-value $Q(s', a')$ for state s' and action a' increases with 3-step SARSA, then $Q(s', a')$ also has to increase with 1-step SARSA.

   [ ] Yes or No [ ]

6. If during one specific episode of 1000 steps, none of the Q-values changes when applying 2-step SARSA, then Q-values would also not change when applying 1-step SARSA.

   [ ] Yes or No [ ]

7. If during one specific episode of 1000 steps, none of the Q-values changes when applying 1-step SARSA, then Q-values would also not change when applying Q-learning.

   [ ] Yes or No [ ]

8. If during one specific episode of 1000 steps, none of the Q-values changes when applying 1-step SARSA, then Q-values would also not change when applying 2-step SARSA.

   [ ] Yes or No [ ]

9. The complexity of the backprop algorithm in a multi-layer deep network scales linearly with the number of synaptic weights

   [ ] Yes or No [ ]

number of correct answers: ...../9

number of wrong answers: ...../9 number of points: ...../ 9

2 Debugging a RL algorithm (9 points)

For debugging of an RL algorithm, you run an agent in a small environment consisting of 8 states with 2 possible actions from each of the 8 states. After running the agent for n episodes, the table of Q-values is as is shown below. For example, we have $Q(7, a1) = 7$.

$a1$	1	2	3	4	5	6	7	8
$a2$	2	4	6	8	6	4	2	6

You run one additional episode and observe the following sequence of several steps, denoted as (s,a,r) = (old state, chosen action, observed reward on next transition)

step 1, $(1, a2, 0)$
step 2, $(2, a2, 2)$
step 3, $(1, a1, 4)$
step 4, $(4, a1, 2)$
step 5, $(5, a1, 0)$
step 6, $(6, a2, 0)$
After step 6 the episode ended.

(a) 3-step SARSA: You work with an implementation of 3-step SARSA on your computer, using a discount factor of $\gamma = 1$ and a learning rate of $\eta = 0.1$.

Question. What are the updated Q- values $Q(2, a2)$ and $Q(1, a1)$ if the algorithm works correctly?

Fill in the blanks. *Example: After the additional episode,* $Q(9, a3)$ *has changed to the new value* $Q(9, a3) = 7.1$.

YOUR ANSWER:

After the additional episode, $Q(1, a1)$ has changes to the new value .....

After the additional episode, $Q(2, a2)$ has changed to the new value .....

number of points: ...../ 2

(b) **1-step Q-learning**: You work with an implementation of 1-step Q-learning on your computer, using a discount factor of $\gamma = 1$ and a learning rate of $\eta = 0.1$.

What are the updated Q-values $Q(2, a2)$ and $Q(1, a1)$ if the algorithm works correctly?

After the additional episode, $Q(1, a1)$ has changed to the new value .....

After the additional episode, $Q(2, a2)$ has changed to the new value .....

number of points: ...../ 2

(c) **SARSA with eligibility traces.** Same scenario as above, but you work with a SARSA algorithm with eligibility traces. Your algorithm calculates the TD error

$\delta \longleftarrow r + \gamma Q(s', a') - Q(s, a)$

with discount factor γ. The decay of an eligibility trace at every step is given by

$e(s_k, a_k) \longleftarrow \beta e(s_k, a_k)$.

[Note that Sutton and Barto write $\beta = \gamma\lambda$ but we work directly with β.]

These traces are then used to update Q-values, with learning rate η.

In your implementation, you are using $\gamma = 1$ and $\beta = 0.9$ and a learning rate of $\eta = 0.1$.

To simplify calculations, you can set $\beta^2 = 0.8$ and $\beta^3 = 0.7$ and $\beta^4 = 0.6$.

After the additional episode, several Q-values will be different.

Calculate the TD errors

Compute the TD error for each and every step and fill the table below.

Step	TD-error
From 1 to 2	
From 2 to 3	
From 3 to 4	
From 4 to 5	
From 5 to 6	
From 6 to the end	

number of points: ...../ 3

What is the new value of $Q(1, a1)$?

After the additional episode, $Q(1, a1)$ takes approximately the new value

.....

number of points: ...../ 1

What is the new value of $Q(2, a2)$?

After the additional episode, $Q(2, a2)$ takes approximately the new value

.....

number of points: ...../ 1

Free space for your calculations, do not use to write down answers.

This page is empty. Free space for your calculations, do not use to write down answers.

3 Gradient descent with Nesterov Momentum and Standard Momentum (12 points + 2 bonus points)

You minimize the loss function

$E(w) = |w| - 1.5$ for $|w| > 1.5$ and $E(w) = 0$ otherwise

with respect to the weight parameter w. To do so you use gradient descent with Nesterov Momentum and you will compare this with Standard Momentum. As a reminder, gradient descent with momentum works with a formula

$\Delta w \longleftarrow -\eta(dE/dw) + \alpha\Delta w.$

The main difference between Nesterov Momentum and Standard Momentum is WHERE the derivative dE/dw is evaluated.

Your friend Anabella claims: *'Nesterov Momentum picks up speed as fast as Standard Momentum when the gradient stays constant, but - because it looks ahead - it is faster in slowing down when the direction of the gradient changes.'* You want to check whether this is true in this example.

In this example, the value of the weight **after** iteration n is called $w(n)$. The initial value before the first update step is $w(0) = 5$**.** You use a learning rate $\eta = 1$. The momentum parameter is $\alpha = 0.8$. For the sake of simplicity, all numerical values are rounded to the first digit after the comma so that $\alpha^2 = 0.6$, $\alpha^3 = 0.5$, $1.4\alpha = 1.1$, $1.8\alpha = 1.4, 1.9\alpha = 1.5, 2.4\alpha = 1.9$ etc. Always use the rounded output of one step as the input of the next step.

(a) What is the value of the weight $w = w^{\text{Stan}}$ after the 3rd, 4th, 5th , and 6th update step using STANDARD momentum? The values after the first and the second update step are already given:

$w^{\text{Stan}}(1) = 4$

$w^{\text{Stan}}(2) = 2.2$

$w^{\text{Stan}}(3) =$.....

$w^{\text{Stan}}(4) =$.....

$w^{\text{Stan}}(5) =$.....

$w^{\text{Stan}}(6) =$.....

number of points: ...../ 2

(b) After how many update steps does w^{Stan} reach the minimum for the first time, when you work with *Standard* momentum?

ANSWER: The time step at which w^{Stand} reaches the minimum is $n_{\text{1st}}^{\text{Stan}}$ =.....

Does w^{Stan} leave the minimum after $n_{\text{1st}}^{\text{Stan}}$?

[ ] Yes or No [ ]
number of points: ...../ 1

(c) Redo an analogous calculation for the NESTEROV momentum. You will find that the first two update steps are identical to those of Standard Momentum.

Where do you evaluate the gradient dE/dw for the 3rd update step using NESTEROV Momentum and what is its gradient?

I need to evaluate the gradient at ..... and find dE/dw = ...

What is the value of the weight $w = w^{\text{Nest}}$ after the third, and after the fourth, update step using NESTEROV momentum?

$w^{\text{Nest}}(3)$ =.....

$w^{\text{Nest}}(4)$ =.....

number of points: ...../ 3

(d) Where do you evaluate the gradient dE/dw for the 5th update step using NESTEROV Momentum and what is its value?

I need to evaluate the gradient at ..... and find dE/dw = ...

What is the value of the weight after the 5th update step?

$w^{\text{Nest}}(5)$ =.....

number of points: ...../ 2

(e) Where do you evaluate the gradient dE/dw for the 6th update step using NESTEROV Momentum and what is its value?

I need to evaluate the gradient at ..... and find dE/dw = ...

What is the value of the weight after the 6th update step?

$w^{\text{Nest}}(6)$ =.....

number of points: ...../ 2

(f) After how many update steps does w^{Nest} reach the minimum for the first time, when you work with *Nesterov* momentum?

ANSWER: The time step at which w^{Nest} reaches the minimum is $n_{\text{1st}}^{\text{Nest}}$ =.....

Does w^{Nest} leave the minimum after $n_{\text{1st}}^{\text{Nest}}$ during the next few update steps?

[ ] Yes or No [ ]

number of points: ...../ 1

(g) THEREFORE, is the statement of your friend Anabella correct in this example?

[ ] Yes or No [ ]

number of points: ...../ 1

(h) **BONUS QUESTION**: What is the final value of w^{Nest} when you work with Nesterov momentum? Start at the position $w^{\text{Nest}}(6)$ that you found before. Do not round the numbers after time step 6, i.e. for calculating $w^{\text{Nest}}(n)$ when $n > 6$.

Write your result first in abstract form using α and then give a numerical value.

Abstract: Using Nesterov momentum, the final value of w^{Nest} is

$\lim_{n\to\infty} w^{\text{Nest}}(n) = w^{\text{Nest}}(6) + f(\alpha)$ with

$f(\alpha)$ =.....

The numerical value is w^{Nest} is $\lim_{n\to\infty} w^{\text{Nest}}(n)$= .....

number of points: ...../ 2

Free space for your calculations, do not use to write down answers.

This page is empty. Free space for your calculations, do not use to write down answers.

4 Automatic (reverse mode) differentiation (12 points)

Our data base $(\mathbf{x}^\mu, \mathbf{t}^\mu)$ has $N = 64$-dimensional input and M-dimensional target output.

We use a quadratic loss function for each pattern μ

$$E(\mu) = (1/2) \sum_{m=1}^{M} [t_m^\mu - \hat{y}_m^\mu]^2 \tag{1}$$

where $\hat{y}_m^\mu$ is the output of unit m in the output layer in response to pattern μ and t_m^μ is the target value for this unit and pattern.

A non-standard pseudo-convolutional neural network can be designed by using sine and cosine nonlinearities in the 1st layer, and using a polynomial nonlinearity that combines sine (k=0) and cosine (k=1) contributions in each localized region of 8 pixels in the next layers. We work here with one spatial dimension. The formal description of the network is shown below.

Hint: the network is sketched on the LAST page (you can detach it if you want).

Output layer

$\hat{y}_m^\mu = \sum_{n=0}^{7} w_{mn}^{(4)} x_n^{(3)}$.

hidden layers (k takes the value 0 or 1).

$x_n^{(3)} = x_{n,0}^{(2)} + x_{n,1}^{(2)}$

$x_{n,k}^{(2)} = (x_{n,k}^{(1)})^4/4$

$x_{n,k}^{(1)} = \sum_{i=0}^{7} \beta_i \sin[\alpha\, x_{8n+i}^{(0)} + k\,(\pi/2)]$.

Input layer: 64 units $x_j^{(0)}$ with $0 \le j \le 63$.

Your task is to derive an efficient* update rule for the parameters α and β_i using stochastic gradient descent on the loss function E.

* efficient means optimal scaling when we change the number of inputs or outputs and optimal re-use of intermediate results.

Parameters $w_{mn}^{(4)}$ are kept fixed through the whole process.

(a) Computational graph (for one pattern)

Given the pattern $\mathbf{x}^\mu$ as the input, draw below the computational graph for going from α and β_i to $E(\mu)$.

Hint 1: for example, if $g(x)$ is a function of a variable x, and if $f(g(x))$ is a function of g, then the computational graph for going from x to f is: $x \to g \to f$.

Hint 2: for example, one node of your graph should be $\hat{y}_m^\mu$, and $\hat{y}_m^\mu$ should be linked to $E(\mu)$ by an arrow or a sequence of arrows.

number of points: ...../ 2

(b) Derivatives for the backward path (for one pattern)

The procedure for the forward path is given by the network formulas on page 12.

For the backward path, for each and every arrow in the computational graph, compute the corresponding derivative as a function of the values of the nodes of the graph, parameters $w_{mn}^{(4)}$, and labels t_m^μ.

Hint 3: for example, for the derivative corresponding to the link mentioned in the Hint 2, you should write $\frac{\partial E(\mu)}{\partial \hat{y}_m^\mu} = -[t_m^\mu - \hat{y}_m^\mu]$.

.....

.....

.....

.....

.....

.....

.....

.....

.....

number of points: ...../ 2

(c) Update rules for one pattern. Write the update of parameters α and β_i as a function of the derivatives identified in the previous part. Indicate bounds for each summation sign.

Hint 4: For example, with respect to the example of Hint 3, your answer should be written in terms of $\frac{\partial E(\mu)}{\partial \hat{y}_m^\mu}$ and not in terms of its specific evaluation $[t_m^\mu - \hat{y}_m^\mu]$.

.....

.....

.....

.....

.....

.....

number of points: ...../ 2

d) Complexity. How does the complexity of the update of the parameter α scale if you double the number of input units (going from 64 to 128, keeping the filter size 8 fixed)?

The algorithm gets slower by a factor of .....

How does the complexity of the update of the parameter β_i scale if you double the number of input units (going from 64 to 128, keeping the filter size 8 fixed)?

The algorithm gets slower by a factor of .....

number of points: ...../ 2

(e) The mistake of Anabella Your friend Anabella did a direct calculation of the gradient of $E(\mu)$ with respect to α and found

$$\frac{dE(\mu)}{d\alpha} = -\sum_{m=1}^{M}[t_m^\mu - \hat{y}_m^\mu] \sum_{k=0}^{1} w_{mn}^{(4)} (x_{n,k}^{(1)})^3 \sum_{i=0}^{7} x_{8n+i}^{(0)} \beta_i \cos[\alpha\, x_{8n+i}^{(0)} + k\,(\pi/2)]$$

She made a mistake in her calculations. What is her mistake? Write down the corrected version exploiting your result from a) - c). Explain why the computational graph and step-wise analysis leading to (c) are useful to avoid the error.

The mistake is

.....

The correct version is

.....

Explanation:

.....

.....

number of points: ...../ 2

(f) Efficient Implementation. You have a total of 9 parameters. The update of each involves many summations. How does an efficient implementation look like (analogous to backprop) for this specific model? Which summations do not need to be repeated 9 times?

.....

.....

.....

number of points: ...../ 2

5 RL theory: Convergence of 2-step Q-learning (12 points)

We have shown in class and the exercises that, assuming that the algorithms converge in expectation, both 1-step and 3-step SARSA converge to a solution of the Bellman equation. Proceed analogously to convince your friend Anabella that the following statement holds:

If the 2-step Q-learning algorithm converges in expectation while playing the max-policy, then it converges to the optimal solution of the Bellman equation.

Please use the following notation: learning rate η; discount factor γ; and denote expectation of a variable ξ by brackets $\langle \xi \rangle$.

It is a single and not very long calculation which you can first do on a scratch paper - but in order to help you structure your thoughts and help us give points for the grading, please write up your solution in the following seven steps.

Step 1. Write down the 2-step Bellman equation for arbitrary policy. Expand the expectations as summations and write your final answer with the notation introduced in the class. Hint: Start with the 1-step Bellman equation and expand it.

.....

.....

For each summation sign, note precisely what we sum over.

number of points: ...../ 2

Step 2. Rewrite the 2-step Bellman equation for the optimal policy.

.....

For each summation sign, note precisely what we sum over. Use notation $a^*(s')$ for the optimal action in state s'.

number of points: ...../ 2

Step 3. Write down the 2-step Q-learning algorithm.

.....

number of points: ...../ 1

Step 4. Rewrite the expected update rule of 2-step Q-learning algorithm with expectation signs (that you will need for the calculations) at the appropriate places:

.....

The expectation signs imply averaging over what? Explain:

.....

.....

number of points: ...../ 2

Step 5. What does the general statement 'has converged in expectation' mean?

Answer: 'A variable y updated by an iterative algorithm A has converged in expectation implies that .....' (complete the sentence)

.....

.....

number of points: ...../ 1

Step 6. Now we exploit convergence in expectation, evaluate the expectations in the result from step 4 and bring it into a form similar to the one in step 2. Be careful to explain what kind of averages/expectations (e.g., if there are summations signs, what are the sums running over?) you evaluate during the calculations. Show two intermediate steps.

.....

.....

.....

number of points: ...../ 3

Step 7. Explain why the proof is finished.

.....

.....

number of points: ...../ 1

This page is empty. Free space for your calculations, do not use to write down answers.

Detachable Figure - illustration of pseudo-Conv-Net in Section 4