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

Artificial Neural Networks: Exam
1st of July 2019

- Write your name in legible letters on top of this page.
- The exam lasts 160 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.
- Put your bag with your computer and smart phone in front of the room, below the blackboard.
- Have your student card displayed before you on your desk.
- **Check that your exam has 16 pages**

Evaluation:

1. ...../6 pts (Quiz-questions)
2. ..... / 11 pts (section 2)
3. ..... / 10 pts (section 3)
4. ..... / 5 pts (section 4)
5. ..... / 6 pts (section 5)
6. ..... / 8 pts (section 6)

Total: ..... / 46 pts

Definitions

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-dimensions: $\mathbf{x}^\mu \in R^N$

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 $\mathbf{x}$.

The Euler number e can be approximated by 2.7.

How to give answers

The first part contains N 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 parts 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 for grading. You can also use colored paper for side calculations.

1 6 Yes-No questions (6 points; explanation on page to the left)

1. A multiplication of the weight vector of the simple perceptron in N dimensions (where N is the input dimension) by a factor of 2 moves the separating hyperplane closer to the origin.

   [ ] Yes or No [ ]

2. The backprop algorithm in a multi-layer deep network scales quadratically with the number of synaptic weights

   [ ] Yes or No [ ]

3. In a two-armed bandit, you initialize Q-values at $Q(a1) = Q(a2) = 5$. The rewards are 0 for action 1 and 3.9 for action 2. Rewards are received immediately, once you have played the action (1-step horizon). With learning rate $\eta = 0.2$ for the iterative update of estimated Q-values and greedy policy, each action is chosen at least 5 times before exploitation begins.

   [ ] Yes or No [ ]

4. We work with a multi-layer networks with rectified linear units in each hidden layer. Is then the following true: 'A cross-entropy error function for multi-class output together with one-hot coding of the target (unit k codes for class k, classes are mutually exclusive) and softmax function for the output plus a sufficiently large number of layers and neurons in each layer is consistent with the idea that the value of the output unit k approximates the posterior probability $P(C_k|\mathbf{x})$, once the optimization algorithm has converged to the minimum over a large set of data points.

   [ ] Yes or No [ ]

5. If we allow for a sufficient number of hidden layers and a sufficient number of units per hidden layer and use 1-hot coding with softmax activation in the output, then a deep convolutional network with trainable weights performs better than other algorithm on all input-output problems where high-dimensional inputs $\mathbf{x}$ are mapped to discrete mutually exclusive classes.

   [ ] Yes or No [ ]

6. You work with a SARSA algorithm and the two actions in state s have estimated Q-values $Q(s, a1) = 20$ and $Q(s, a2) = 21$. Under a softmax policy with $\beta = 1$, action a1 is chosen with probability less than 30 percent.

   [ ] Yes or No [ ]

Number of correct answers: ...../6

Number of wrong answers: ...../6 number of points: ...../ 6

2 Debugging a RL algorithm (11 points)

For debugging of an RL algorithm you construct a small environment. You work with 8 states and have 2 possible actions from each of the 8 states. You read-out the table of Q-values after n episodes and find the following values:

$Q(1, a1) = 2$, $Q(2, a1) = 2$, $Q(3, a1) = 3$, $Q(4, a1) = 6$, $Q(5, a1) = 6$, $Q(6, a1) = 12$, $Q(7, a1) = 10$, $Q(8, a1) = 9$,

$Q(1, a2) = 1$, $Q(2, a2) = 1$, $Q(3, a2) = 2$, $Q(4, a2) = 2$, $Q(5, a2) = 4$, $Q(6, a2) = 4$, $Q(7, a2) = 4$, $Q(8, a2) = 6$,

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, $(1, a2, 1)$
step 2, $(2, a2, 1)$
step 3, $(3, a1, 0)$
step 4, $(5, a1, 4)$
step 5, $(6, a1, 1)$
step 6, $(8, a2, 1)$

(a) 2-step SARSA: You work with an implementation of 2-step SARSA on your computer, using a discount factor of $\gamma = 1/3$ and a learning rate of $\eta = 1/2$.

Question. What are the updated Q- values $Q(1, a2)$ and $Q(3, a1)$ if the algorithm works correctly?

Fill in the blanks. *Example: After the additional episode,* $Q(9, a3)$ *has changed to the new value* $Q(9, a3) = 7/5$.

YOUR ANSWER:

After the additional episode, $Q(1, a2)$ has changed to the new value .....

After the additional episode, $Q(3, a1)$ has changes 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 $\gamma = 1/3$ and with a learning rate of $\eta = 1/2$.

What are the updated Q- values $Q(1, a2)$ and $Q(3, a1)$ if the algorithm works correctly?

After the additional episode, $Q(1, a2)$ has changed to the new value .....

After the additional episode, $Q(3, a1)$ has changed to the new value .....

number of points: ...../ 2

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

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

with discount factor γ. The decay of an eligibility trace 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 $\gamma = \beta = 1/3$ and a learning rate of $\eta = 1/2$.

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

What is the new value of $Q(1, a2)$? To simplify calculations, you can set the eligibility trace to zero if it has a value smaller or equal to $\beta^3 = 1/27$.

After the additional episode, $Q(1, a2)$ takes approximately the new value

.....

number of points: ...../ 2

What is the new value of $Q(3, a1)$? To simplify calculations, you can set the eligibility trace to zero if it has a value smaller or equal to $\beta^3 = 1/27$.

After the additional episode, $Q(3, a1)$ takes approximately the new value

.....

number of points: ...../ 2

(d) **Do the three algorithms in (a), (b), (c) predict the same direction of change (same sign) or the same value for the two Q-values?**

.....

.....

number of points: ...../ 1

Do you expect that this observation holds for all RL-problems? Why?

.....

.....

number of points: ...../ 1

(e) **Which one of the above three algorithms is a TD algorithm? Which one is not? Justify your answer.**

.....

.....

number of points: ...../ 1

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

3 BackProp with exponential units (10 points)

Our data base $(\mathbf{x}^\mu, \mathbf{t}^\mu)$ has N-dimensional input and N-dimensional target output. We have a fully connected network with two hidden layers each of size N.

We use a quadratic loss function for each pattern μ

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

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.

Neurons in the hidden layers and output layer have an exponentional activation function.

$g(a_i^l) = \exp(a_i^l)$

with $a_i^l = \beta_i^l \left(\sum_j w_{ij}^l x_j^{l-1} - 1 \right)$. The index $1 \leq l \leq 3$ indicates the layer. The input units have a $l = 0$. Note that, in contrast to standard neural networks the neurons have no threshold parameter but a fixed threshold of value 1. Instead there is a multiplier β_i^l in front of the sum.

We consider the weights w_{ij}^l as fixed. (As an example, you may think of the weights as implementing given convolutional filters.)

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

* efficient means optimal scaling when we change the number of neurons

Summarize your results in pseudo-code using the layout on the next page:

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

Pseudocode (for one pattern)

(0) Initialization

Parameters w_{ij}^l are loaded and kept fixed thereafter.

Parameters β_i^l are initialized at small values $[\epsilon, \epsilon]$.

Pattern $\mathbf{x}^\mu$ is applied at the input.

(a) Forward pass

Evaluate/iterate/loop over

.....

.....

.....

I need to store the following variables for later use:

.....

.....

number of points: ...../ 2

(b) Backward pass

Evaluate/iterate/loop over

.....

.....

.....

.....

.....

I need to store the following variables for later use:

.....

.....

number of points: ...../ 3

(c) Update of parameters β_i^l

Evaluate/iterate/loop over

.....

.....

number of points: ...../ 3

(d) The network has two hidden layers with $K = N$ nonlinear neurons in each hidden layer. How does your algorithm scale if you increase the number K of neurons per hidden layer by a factor of 100 to a new value $K = 100N$? Justify your answer by referring to your results in (a) - (c).

If I increase the number of hidden neurons by a factor 100, the algorithm is expected to take longer by a factor of (approximately) .....

because .....

.....

number of points: ...../ 2

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

4 Dropout (5 points)

We have a deep network of $2n$ hidden layers ($n > 2$) of neurons with sharp threshold functions $g(a) = 1$ for $a > 0$ and zero otherwise. After training with dropout, somewhere in hidden layer n, we have a hidden neuron i which receives input from 4 hidden neurons in layer $n - 1$.

All weights onto neuron i are equal to one and the threshold of neuron i is 2.7.

Each of the four hidden neurons j in layer $n - 1$ receives input from the same 2 neurons in layer $n - 2$. The weight vectors and thresholds of the four neurons in layer $n - 1$ are:

j=1 (1,0) and threshold 0
j=2 (1,0) and threshold 0.5
j=3 (1,1) and threshold 1
j=4 (1,-1)and threshold 1

(i) Qualitatively sketch the two-dimensional space representing the activity of the 2 neurons in layer $n - 2$ and indicate the region (by shading it with crosses x x x) in which neuron i responds positively.

Sketch here. number of points: ...../ 2

(ii) Dropout: Remove neurons $j = 1$ and $j = 4$ in layer $n - 1$, rescale the weights appropriately, and sketch the input space where neuron i responds positively (by shading it with crosses x x x).

Sketch here. number of points: ...../ 1

(iii) Dropout: Remove neurons $j = 2$ and $j = 3$ in layer $n - 1$, rescale the weights appropriately, and sketch the input space where neuron i responds positively (by shading it with crosses x x x).

Sketch here. number of points: ...../ 1

(iv) Your friend Adam claims: 'Dropout might be a useful trick, but nobody understands how it works'. Your friend Berthilde claims 'Dropout is good for generalization and easy to understand'. Comment on your results (think also of the other 4 combinations of dropping out two neurons) and relate your results to the claims of your friends.

.....

.....

.....

.....

number of points: ...../ 1

5 RL theory: Convergence of 2-step SARSA (6 points)

We have shown in class and the exercises that, assuming that the algos 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 assuming that the 2-step SARSA algo converges in expectation, it converges to a 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 four steps.

Step 1. Write down the 2-step SARSA algorithm and THEN (second line) add the 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 2. What does the assumption 'the algorithm has converged in expectation' mean?

Answer: The algorithm has converged in expectation' implies

.....

.....

(Don't evaluate the expectations yet)

number of points: ...../ 1

Step 3. Start now from the other side: Take the one-step Bellman equation and expand it so as to arrive at a two-step equation.

If we start from the 1-step Bellman equation, we find with two steps of calculation.

.....

.....

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

number of points: ...../ 1

Step 4: Now we evaluate the expectations in the result from steps 1 and 2 and bring it into a form similar to the one in step 3. Be carefull to explain what kind of averages/expectations (e.g., if there are summations signs, what are the sums running over?) you evaluate curing the calculations.

.....

.....

.....

.....

Explain why the proof is finished.

.....

number of points: ...../ 2

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

6 RL theory: Loss-function for 2-step SARSA (8 points)

You have a task with 4 actions. Each episode lasts for more than 2 time steps (e.g., 7 time steps). You use a network with a single layer of 1000 hidden neurons and 4 output neurons representing the Q-values $Q(a_1|s), ...Q(a_4|s)$ for the 4 actions. The state space is continuous and the state is denoted by s while the state at time t is denoted by s_t.

The outputs are

$Q(a_i|s) = \sum_k w_{ik}\phi_k(s)$

where ϕ_k is the activity of hidden neuron k.

(i) If you want to implement a 2-step SARSA in a neural network, you need to define a loss function that you optimize with respect to the network parameters. suppose you have run a 'batch' of 10 episodes of 7 time steps each. Write down the loss function for 2-step SARSA which you want to optimize. Don't forget to put summation signs, whenever you need them. [You can assume implicitly that Q-values for $t > 7$ are set to zero, so as to simplify the notation].

.....

.....

number of points: ...../ 1

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

(ii) Assume now that the hidden layer of the network implements non-overlapping discrete indicator functions, i.e., for each possible state s exactly one hidden neuron is activated at a rate $\phi_k(s) = 1$ and all others are zero. Thus a hidden neuron k can be identified with a state s_k.

Calculate the stochastic online gradient update rule for the weight w_{23} by taking the **full derivative** of the loss function in (ii), and then going from batch to online:

The result of the calcuation is

$\Delta w_{23} = \eta$.....

.....

With a stochastic online gradient update rule of the above error function, how much does the weight w_{23} change if, at time $t = 5$, you take action 4 in state 2 and at times $t + 1$ and $t + 2$ you take action 4 in states 5 and 6?

$\Delta w_{23} = \eta$.....

How much does it change if at time $t = 5$, you take action 3 in state 2 and at times $t + 1$ and $t + 2$ you take action 4 in states 5 and 6?

$\Delta w_{23} = \eta$.....

How much does it change if you take at time $t = 5$, action 2 in state 3 and at times $t + 1$ and $t + 2$ you take action 4 in states 5 and 6?

$\Delta w_{23} = \eta$.....

How much does it change if you take at time $t = 5$, action 4 in state 3 and at times $t + 1$ and $t + 2$ you take action 2 in states 1 and 3?

$\Delta w_{23} = \eta$.....

number of points: ...../ 5

(iii) Is the learning rule that you found by stochastic gradient descent the standard tabular 2-step SARSA rule?

[ ] yes, the learning rule found by stochastic gradient update is identical to standard tabular 2-step SARSA.

[ ] no, the learning rule found by stochastic gradient update is not identical to standard tabular 2-step SARSA.

Justify your answer. What is the 'tricky' aspect one should keep in mind? (You can use terms such as 'gradient', 'error', 'target', 'semi-gradient', 'slowly-changing' etc).

.....

.....

.....

.....

number of points: ...../ 2