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

Artificial Neural Networks: Exam
5th of July 2021

- 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, doublesided**.
- No calculator is allowed.
- Have your student card displayed in front of you on your desk.
- **Check that your exam has 13 pages; page 14 is an empty backcover**

Evaluation:

1. ..... / 10 pts (Section 1, Quiz-questions)
2. ..... / 5 pts (Section 2, Bellman equation)
3. ..... / 14 pts (Section 3, Gradient Descent)
4. ..... / 6 pts (Section 4, Policy Gradient)

Total: ..... / 35 pts

This page remains empty. You can use it as free space for your calculations, do not use to write down answers.

Definitions and notations

RL stands for Reinforcement Learning.

The symbol η is reserved for the learning rate.

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

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 state space is continuous then states 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 ask for extra scratch paper. We will not look at the scratch paper.

1 10 Yes-No questions (10 points; explanation on the previous page)

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

   [ ] Yes or No [ ]

2. 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 [ ]

3. For a simple network with a single output (with strictly monotone nonlinearity) and no hidden layer, the stochastic gradient descent update rule on an error function $E = \sum_\mu [t^\mu - \hat{y}^\mu]^2$ 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 [ ]

4. For a simple Perceptron with input $\mathbf{x}$ in N dimensions, the following holds: if in the N dimensional picture the threshold is changed, then this induces a rotation of the hyperplane in the $N+1$ dimensional picture.

   [ ] Yes or No [ ]

5. In state s we have three action choices with Q-values $Q(s,a_1) = 2$, $Q(s,a_2) = 4$, $Q(s,a_3) = 5$ and use the softmax policy. If I double all Q-values (i.e., $Q(s,a_1) = 4$, $Q(s,a_2) = 8$, $Q(s,a_3) = 10$), then the relative preference for action a_3 increases.

   [ ] Yes or No [ ]

6. You use an error function E that depends on parameters $\mathbf{w}$, minimize the error on the training set, and evaluate on the validation set which yields a value E^*. You then switch to a cost function $C(\mathbf{w}) = E(\mathbf{w}) + \lambda|\mathbf{w}|$ where the absolute value signs denote the L2-norm and $\lambda > 0$, mininize the cost on the training set, and evaluate on the validation set which yields C^*. Since λ is positive and absolute values are positive, C^* is always larger than E^*.

   [ ] Yes or No [ ]

The next four yes-no questions all have the same conditions: In an RL problem, the agent moves in a directed state-transition graph, i.e., during an episode (one complete path) a state is touched at most once. The agent starts in state s_1, chooses between two actions in each state, and arrives after 10 actions in a terminal state s_{term}. In the first episode the agent receives after the fifth action a reward of $r = +1$. This is the only reward on the path of the first episode; all other rewards are zero.

1. The agent uses 2-step SARSA with a discount factor of $\gamma = 0.8$ and all Q-values are initialzed at $Q(s, a) = +2$. Then exactly two Q-values will have increased after the first episode.

   [ ] Yes or No [ ]

2. Same conditions as before: The agent uses 2-step SARSA with a discount factor of $\gamma = 0.8$ and all Q-values are initialzed at $Q(s, a) = +2$. Then at least one Q-value will have decreased after the first episode.

   [ ] Yes or No [ ]

3. The agent uses an (advantage) actor-critic with eligibility traces for both actor and critic. Eligibility traces decay with $\lambda = 0.6$. Reward estimators are described by V-values initialized at $V(s) = +2$ for all states including the terminal state. The discount factor for V-values is one. Then at least three V-values will have increased after the first episode.

   [ ] Yes or No [ ]

4. The agent uses an (advantage) actor-critic with eligibility traces for both actor and critic. Eligibility traces decay with $\lambda = 0.6$. Reward estimators are described by V-values initialized at $V(s) = +2$ for all states including the terminal state. The discount factor for V-values is one. Then at least three V-values will have decreased after the first episode.

   [ ] Yes or No [ ]

number of correct answers: ...../10

number of wrong answers: ...../10 number of points: ...../ 10

This page remains empty. You can use it as free space for your calculations, do not use to write down answers.

2 Bellman equation for a new algorithm (5 points)

In class we have seen that algorithms such as SARSA can be linked to a Bellman equation. Now your friend Thomas proposes a new variant of SARSA (which he calls '1-2-3-SARSA') with the update rule for pseudo-Q-values $\tilde{Q}$.

$$\Delta\tilde{Q}(s_n, a_n) = \eta[1 \cdot r(s_n, a_n) + 2 \cdot \tilde{Q}(s_{n+1}, a_{n+1}) - 3 \cdot \tilde{Q}(s_n, a_n)] \tag{1}$$

where s_n denotes the state encountered in the nth step of the episode, a_n the action taken in the nth step of the episode, and $r(s_n, a_n)$ the reward received after this state-action pair and $\eta < 0.01$ is a parameter.

(a) What is the interpretation of $\tilde{Q}$? Is it qualitatively different from standard Q-values or is there are one-to-one relation between $\tilde{Q}$ and the Q-values found by a normal SARSA algorithm?

.....

.....

.....

number of points: ...../ 2

(b) Suppose that this rule has converged in expectation. What is the resulting Bellman equation for $\tilde{Q}$? Give the full Bellman equation for $\tilde{Q}$ assuming finite sets of states and actions and a probabilistic transition matrix.

.....

.....

.....

number of points: ...../ 3

3 BackProp for event-based neural networks (14 points)

Our data base contains entries $(\mathbf{x}^\mu, \mathbf{y}^\mu)$ where $\mathbf{x}^\mu$ is a J-dimensional input vector with elements $0 \leq x_j^\mu \leq 1$ (where $1 \leq j \leq J$) and a K-dimensional target output with elements $0 \leq y_k^\mu \leq 1$ (where $1 \leq k \leq K$). We approximate the relation between input and output using a fully connected network with three hidden layers, each of size N. The index of the entries runs from $\mu = 1$ to $\mu = 10000$.

Input, output, and hidden layers use neurons that code information in the timing of 'events' (sometimes called 'spikes'). Inputs during presentation of pattern μ are encoded in event times (spike times) $t_j^0 = x_j^\mu$ where the upper index zero indicates layer zero (= input layer) and $1 \leq j \leq J$ (see Figure on next page). Outputs are encoded in the event times t_k^4 of neurons in the output layer with layer index $l = 4$ and $1 \leq k \leq K$.

The aim is the minimize the quadratic loss function for each pattern μ

$$E(\mu) = \frac{1}{2} \sum_{k=1}^{K} [t_k^4 - y_k^\mu]^2 \tag{2}$$

where t_k^4 is the event timing of unit k in the output layer in response to pattern μ and y_k^μ is the target value for this unit and pattern.

The event-based model consists of layers $0 \leq l \leq 4$ of spiking neurons. The variable v_p^l of neuron p in layer $l \geq 1$ evolves as a function of time as

$$v_p^l(t) = \alpha t + \sum_i w_{pi}^l \cdot (t - t_i^{l-1}) \cdot H(t - t_i^{l-1}) \tag{3}$$

where $t \geq 0$ is time and H denotes the Heaviside step function with $H(x) = 1$ for $x > 1$ and zero otherwise. Here α is the slope of increase in the absence of inputs. The slope changes after each input event. Note that only events that occur BEFORE t can contribute to the slope at time t (see Figure).

The process is stopped after an observation time t^{obs} and only events that occur before t^{obs} are taken into account. The event time t_p^l of neuron p in layer l follows from the threshold condition $v_p^l = \vartheta$ and is given implicitly by

$$t_p^l = \frac{\vartheta + \sum_i w_{pi}^l t_i^{l-1} H(t_p^l - t_i^{l-1})}{\alpha + \sum_i w_{pi}^l H(t_p^l - t_i^{l-1})} H(t^{obs} - t_p^l) + (1 + \epsilon) t^{obs} \, H(t_p^1 - t^{obs}) \tag{4}$$

Note that events that would occur after the observation time are excluded because of the Heaviside function $H(t^{obs} - t_p^l)$.

The derivative of the loss for pattern μ with respect to a weight w_{ij}^1 in the first layer is

$$\frac{\partial E}{\partial w_{ij}^1} = -\sum_{k=1}^{K}\sum_{n=1}^{N}\sum_{m=1}^{N}[t_k^4 - y_k^\mu] \tag{5}$$

$$\cdot\frac{w_{kn}^4}{v'(t_k^4)}\cdot\frac{w_{nm}^3}{v'(t_n^3)}\cdot\frac{w_{mi}^2}{v'(t_m^2)}\cdot\frac{t_i^1 - t_j^0}{v'(t_i^1)}$$

$$\cdot H(t^{obs} - t_k^4)H(t^{obs} - t_n^3)H(t^{obs} - t_m^2)H(t^{obs} - t_i^1)$$

$$\cdot H(t_k^4 - t_n^3)H(t_n^3 - t_m^2)H(t_m^2 - t_i^1)H(t_i^1 - t_j^0) \tag{6}$$

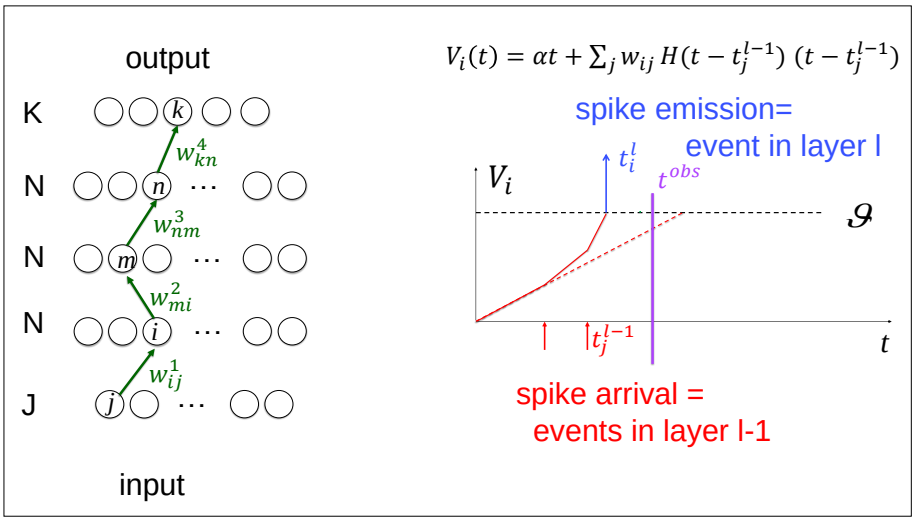
Here $v'(t_p^l) = \alpha + \sum_i w_{pi}^l \cdot H(t_p^l - t_i^{l-1})$ denotes the derivative of the variable v_p^l (rising slope) at the moment of threshold crossing. We assume that weights are always such that the slope is positive at all moments (and in particular at the moment of threshold crossing).

Your task is to derive an efficient* backprop rule to update all the weights w_{ij}^1 in the first layer. To do so, indicate all varieables that you need to store during the forward and backward pass and how they are used in the weight update step.

* efficient means optimal scaling when we change the number N of neurons in the hidden layers.

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

SCHEMATIC IMAGE: Left: A network with three hidden layers. Indices of selected weights and neurons are given. RIGHT: An event-based unit. Events of layer l-1 change the slope. The event in layer l is defined by the moment of threshold crossing.



Pseudocode (for one pattern)

(0) Initialization

Parameters w_{ij}^l are initialized

Parameters $\alpha > 0, \vartheta > 0$ are initialized at small values $[\epsilon, \epsilon]$.

Pattern $\mathbf{x}^\mu$ is applied at the input (using the event-based code explained above)

(a) Forward pass

Evaluate and store the following variables in the following order

.....

.....

.....

.....

.....

.....

.....

number of points: ...../ 2

(b) Backward pass

Evaluate and store the following variables in the following order

.....

.....

.....

.....

.....

.....

.....

.....

number of points: ...../ 4

(c) Update of weights

$\Delta w_{ij}^1 =$ .....

number of points: ...../ 4

(d) Suppose that the slope parameter α is also a free parameter. Update the slope α_i^1 of neuron i in the first layer using gradient descent

$\Delta\alpha_i^1 =$ .....

number of points: ...../ 4

(d) The network has three hidden layers with N neurons in each hidden layer. How does your algorithm scale if you increase the number N of neurons per hidden layer from $N = 10$ to $N = 200$? Justify your answer by referring to your results in (a) - (c) and exploit that the number of neurons in the input and the output layer do not change.

ANSWER:

If I increase the number of hidden neurons by a factor 20, the algorithm is expected to take longer

by a factor of (approximately) .....

because

.....

.....

number of points: ...../ 2

4 Policy gradient (8 points)

An agent moves in a 2-dimensional discrete environment. In each episode, it starts (with probability $P(\mathbf{x}_n)$) from one of 9900 possible initial states $\mathbf{x}_n$ with $1 \leq n \leq 99000$. In each initial state it has two possible actions a_k with $k \in \{1, 2\}$. The action choices are implemented by an artificial neural network with an 'action-related' neuron which emits one of two output values a_1, either 0 or 1. The probability that this neuron emits a value $a_1 = 1$ is

$$\pi(a_1 = 1|\mathbf{x}; \mathbf{w}) = \frac{[\sum_{k=1}^{2500} w_k y_k]^4}{1 + [\sum_{k=1}^{2500} w_k y_k]^4} \tag{7}$$

where $\mathbf{w}$ is the vector of weights w_k and $y_k = f(\mathbf{x} - \mathbf{c}_k) \geq 0$. Here $\mathbf{x} \in \mathbb{R}^2$ is the input signal that contains the state information and $\mathbf{c}_k$ is the center of one of 2500 localized but overlapping basis functions f that fully cover the two-dimensional input space. The centers lie on a 50x50 grid and initial states are either at the center of the grid or in between two grid points.

If $a_1 = +1$ then the agent chooses action 1 which makes him move North with probability 0.5, or East or West with probability 0.25 each. If $a_1 = +0$ then the agent chooses action a_2 with probability $1 - \pi(a_1 = 1|\mathbf{x}; \mathbf{w})$. The second action choice makes him moves by South with probability 0.5, or East or West with probability 0.25 each. For an action a_i in state $\mathbf{x}_n$, the agent receives a reward $R(a_i, \mathbf{x}_n)$ and then the agent is reinitialized in a new state. Each movement is by half a grid length.

(a) Calculate the gradient of the mean reward

$\langle R \rangle = \sum_{n=1}^{99000} \sum_{i=1}^{2} R(a_i, \mathbf{x}_n) \pi(a_i|\mathbf{x}_n; W) P(\mathbf{x}_n)$ with respect to the weight w_{i7}.

$\frac{d}{dw_{i7}} \langle R \rangle =$

.....
.....
.....
.....
.....

number of points: ...../ 3

(b) Use the result from (a) to write down a 'batch policy gradient' rule for an arbitrary weight w_{ij}.

$\Delta w_{ij} =$

.....

number of points: ...../ 1

(c) Go from the batch rule for weight w_{ij} in (b) to an 'online rule' that would allow the agent to update the weight after each action.

$\Delta w_{ij} =$

.....

number of points: ...../ 2

The rest of this page is empty. Free space for your calculations, do not use to write down answers.