Wendelin Böhmer <j.w.bohmer@tudelft.nl>

Math and machine learning primer

All assignments (questions marked with an A that yield points) must be submitted on Brightspace before the corresponding tutorial begins (see due date). Please submit all answers in one PDF file. Scans of handwritten solutions (e.g. for math answers) are permitted, but must be readable and have a file-size below 5MB. Do not submit the voluntary exercises (marked with an E), which will not earn you any points, but can help you practice the math and prepare for the exam.

You will also be asked to implement and test something in python/pytorch. We recommend that you use a Jupyter Notebook, convert your final version (including result plots) to PDF ("Download as" → "PDF via LaTeX") and attach the PDF at the end of your submission PDF. You are welcome to use other editors, but please make sure you submit the code (or the crucial code segments) and the results in an easily readable format together with your theory answers as one PDF file.

A sample solution will be published after the tutorial. Please always demonstrate how you arrived at your solution. You will receive the points when you convince us that you have seriously attempted to answer the question, even if your answer is wrong. You qualify for the exam when you have earned 75% of the total achievable points (sum over all 4 exercise sheets, not for individual sheets).

Good luck!

A1.1: Implement MNIST classification **(5 points)**

Implement the MNIST classification example from the lecture slides. Make sure you get the correct deep CNN model architecture from Lecture 2 (p.18).

(a) *[2 points]* Train the model $\boldsymbol{f}_\theta : \mathbb{R}^{28\times 28} \to \mathbb{R}^{10}$ from the lecture slides with a cross-entropy loss for 10 epochs. Plot the average train/test losses during each epoch (y-axis) over all epochs (x-axis). Do the same with the average train/test accuracies during each epoch. Try to program as modular as possible, as you will re-use the code later.

(b) *[1 point]* Change your optimization criterion to a mean-squared-error loss between the same model architecture $\boldsymbol{f}_\theta : \mathbb{R}^{28\times 28} \to \mathbb{R}^{10}$ you used in (a) and a one-hot encoding ($\boldsymbol{h}_i \in \mathbb{R}^{10}, h_{ij} = 1$ iff $j = y_i$, otherwise $h_{ij} = 0$) of the labels y_i:

$$\mathcal{L} \quad := \quad \frac{1}{n}\sum_{i=1}^{n}\Big(\boldsymbol{f}_\theta(\boldsymbol{x}_i) - \boldsymbol{h}_i\Big)^2$$

Plot the same plots as in (a). Try to reuse as much of your old code as possible, e.g., by defining the criterion (which is now different) as external functions that can be overwritten.

(c) *[1 point]* Define a new architecture $f'_\theta : \mathbb{R}^{28\times 28} \to \mathbb{R}$, that is exactly the same as above, but with only one output neuron instead of 10. Train it with a regression mean-squared error loss between the model output and the scalar class identifier.

$$\mathcal{L}' \quad := \quad \frac{1}{n}\sum_{i=1}^{n}\Big(f'_\theta(\boldsymbol{x}_i) - y_i\Big)^2$$

Plot the same plots as in (a), but for 50 epochs.

(d) *[1 point]* Learning in (c) should be significantly slower, in terms of accuracy gain per epoch, than in (a) and (b). Use a transformation of your model output (which can be implemented in the functions that compute the criterion and the accuracy, or as an extra module) as $f''_\theta(\boldsymbol{x}_i) := \alpha f'_\theta(\boldsymbol{x}_i) + \beta$, with $\alpha = \beta = 4.5$. Plot the same plots as in (c). Does the learning behavior change? Why?

Bonus-question: Can you come up with an alternative approach to (d) that has the same speed-up effect?

Hint: Evaluate your test loss and accuracy before every training to make sure the accuracy is defined correctly (should be around 0.1 for a model without training). This means that you will always have one test measurement more.

Hint: Try to reuse as much of your old code as possible, e.g., by defining the criterion and the accuracy (which will change for some question) as external functions that can be overwritten later.

A1.2: Mean and variance of online estimates **(3 points)**

Let $\{y_t\}_{t=1}^{\infty}$ an infinite training set drawn i.i.d. from the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. At time t, the online estimate f_t of the average over the training set, starting at f_0, is defined as

$$f_t \quad = \quad f_{t-1} \quad + \quad \alpha\,(y_t - f_{t-1})\,, \qquad\qquad 0 < \alpha < 1\,.$$

(a) *[1 point]* Show that for small t the online estimate is *biased*, i.e., $\mathbb{E}[f_t] \neq \mu$.

(b) *[1 point]* Prove that in the limit $t \to \infty$ the online estimate is unbiased, i.e., $\mathbb{E}[f_t] = \mu$.

(c) *[1 point]* Prove that in the limit $t \to \infty$ the variance of the online estimate is $\mathbb{E}[f_t^2] - \mathbb{E}[f_t]^2 = \frac{\alpha\,\sigma^2}{2-\alpha}$.

Hint: You can use the geometric series $\sum_{k=0}^{t-1} r^k = \frac{1-r^t}{1-r},\ \forall |r| < 1\,.$

Bonus-question: Prove that for the decaying learning rate $\alpha_t = \frac{\alpha}{1-(1-\alpha)^t}$ holds $\lim_{\alpha \to 0} \alpha_t = \frac{1}{t}$.

Hint: You can also use the binomial identity $(x+y)^t = \sum_{k=0}^{t} \binom{t}{k} x^k\, y^{t-k}$.

A1.3: Noise in linear functions **(4 points)**

Let $\{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^m$ denote a set of training samples and $\{y_i\}_{i=1}^n \subset \mathbb{R}$ the set of corresponding training labels. We will use the mean squared loss $\mathcal{L} := \frac{1}{n}\sum_i (f(\boldsymbol{x}_i) - y_i)^2$ to learn a function $f(\boldsymbol{x}_i) \approx y_i, \forall i$.

(a) *[1 point]* Derive the analytical solution for parameter $\boldsymbol{a} \in \mathbb{R}^m$ of a linear function $f(\boldsymbol{x}) := \boldsymbol{a}^\top \boldsymbol{x}$.

(b) *[1 point]* We will now augment the training data by adding i.i.d. noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2) \in \mathbb{R}$ to the training labels, i.e. $\tilde{y}_i := y_i + \epsilon_i$. Show that this does not change the analytical solution of the expected loss $\mathbb{E}[\mathcal{L}]$.

(c) *[1 point]* Let f denote the function that minimizes $\mathcal{L}$ *without* label-noise, and let $\tilde{f}$ denote the function that minimizes $\mathcal{L}$ *with* a random noise ϵ_i added to labels y_i (but *not* the solution of the expected loss $\mathbb{E}[\mathcal{L}]$). Derive the analytical variance $\mathbb{E}[(\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x}))^2]$ of the noisy solution $\tilde{f}$.

(d) *[1 point]* We will now augment the training data by adding i.i.d. noise $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \in \mathbb{R}^m$ to the training samples: $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{\epsilon}_i$. Derive the analytical solution for parameter $\boldsymbol{a} \in \mathbb{R}^m$ that minimizes the expected loss $\mathbb{E}[\mathcal{L}]$.

Bonus-question: Which popular regularization method is equivalent to (d) and what problem is solved?

Hint: Summarize all training samples into matrix $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times m}$, all training labels into vector $\boldsymbol{y} = [y_1, \ldots, y_n]^\top \in \mathbb{R}^n$, and denote the noisy versions $\tilde{\boldsymbol{y}} \in \mathbb{R}^n$ and $\tilde{\mathbf{X}} \in \mathbb{R}^{n \times m}$.

A1.4: Discounted values can be counter intuitive **(3 points)**

When computer scientists design the tasks that deep RL algorithms are supposed to solve, they often assume the algorithm would find the policy that collects the most reward, e.g. maximizes the average reward. However, most deep RL algorithms are based on discounted values and do not optimize this measure directly, which can yield counter-intuitive results. Take the following MDP[1] that has only one decision state s_0 with the choice $\mathcal{A} := \{L, R\}$. If the agent chooses $L \in \mathcal{A}$, it receives the reward r and embarks on a journey of n states $s_1, \dots, s_n, s_0$, during which it cannot make any decisions (or all decisions have the same outcome). If on the other hand the agent chooses $R \in \mathcal{A}$ in s_0, it receives no immediate reward and embarks on a different journey of n states $s'_1, \dots, s'_n, s_0$, during which it also cannot make a decision, but which *ends* with a reward of r' for the transition $s'_n \to s_0$. Note that neither choice ends the Markov chain, which has therefore infinite length.

(a) Derive the discounted infinite-horizon value $V^\pi(s_0)$ for one policy that chooses $\pi(L|s_0) = 1$ and for another policy that chooses $\pi'(R|s_0) = 1$. Give a solution without infinite sums if you can.

(b) For $r < r'$, derive the range of $\gamma \in [0, 1)$ for which the optimal policy based on the discounted value does **not** also yield the largest *average reward*.

A1.5: Expected training labels minimize classification loss **(2 points)**

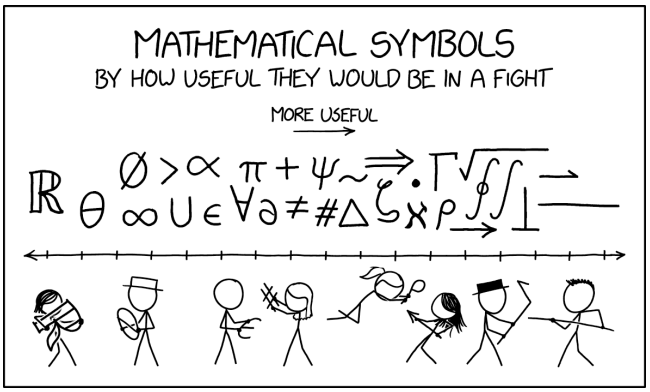
In this exercise we would like to correctly predict the probability $\pi_\theta(a|s)$, parameterized by θ, of 'optimal actions' $a \in \mathcal{A} \subset \mathbb{N}$ given some states $s \in \mathcal{S} \subset \mathbb{R}^d$. The training-set is drawn from state distribution $p(s)$ and stochastic training policy $p(a|s)$. As this is essentially a classification task, we use the *negative log-likelihood* as loss function:

$$\mathcal{L}[\theta] \quad := \quad -\int p(s) \sum_{a \in \mathcal{A}} p(a|s) \log \pi_\theta(a|s)\, ds\,.$$

Prove analytically that the policy $\pi_\theta(a|s) = p(a|s), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$, is an extreme point of this loss.

Bonus-question: Can you also show that it is a minimum?

Hint: Gradient operators ∇ pass through sums and integrals, e.g. $\nabla \sum_i f_i(x) = \sum_i \nabla f_i(x)$, and the gradient of the logarithm is $\nabla \log f(x) = \frac{\nabla f(x)}{f(x)}$. Minima have a positive definite Hessian matrix. You can look up those terms at `https://en.wikipedia.org/wiki/Definite_matrix` and `https://en.wikipedia.org/wiki/Hessian_matrix`.



`https://xkcd.com/2343`

Total 17 points.

[1] This MDP is based on the counter-example from Naik et al. (2019) ⟨`https://arxiv.org/abs/1910.02140`⟩.