Statistical Machine Learning – Homework

Prof. Dr. Jan Peters
Daniel Palenicek, An Thai Le, Theo Gruner, Maximilian Tölle & Théo Vincent

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 2023 – Due date: 14.07.2023, 23:00
Sheet 4

Each (sub)task in this homework is worth 1 point. For example, you can get up to 5 points in Task 4.1. The points you achieve in this homework will count towards the exam bonus points.
The solutions need to be uploaded to Moodle before the deadline. You can provide us with scans of your handwritten solutions or directly write them into our .tex file and submit them as .pdf. Please make sure that we can determine exactly which solution belongs to which (sub)task. If you decide on handwritten solutions, please understand that we can only give points for answers which we can also read!

Neural Networks from scratch

Task 4.1: Background Theory on SoftMax, CrossEntropy and Tanh

In this task, we want you to derive the equations needed to implement important parts of you neural network. We will use the neural network for handwritten digit recognition (classification) and therefore use the softmax activation in combination with a cross-entropy loss. We use the Tanh as a non-linear activation function to create a deeper neural network.

4.1a)

Let $\mathbf{x} \in \mathbb{R}^{1\times C}$ and $\boldsymbol{\sigma}(\mathbf{x}) = \text{softmax}(\mathbf{x}) = [\sigma_i(\mathbf{x})]_{i=1,\dots,C}$ where $\sigma_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$.
Derive the matrix expression for the softmax's Jacobian $J_{\mathbf{x}}\boldsymbol{\sigma}(\mathbf{x}) \in \mathbb{R}^{C\times C}$ which is formulated only in terms of the softmax outputs $\sigma_i(\mathbf{x})$.

Solution:

4.1b)

The backward propagation function of the softmax computes the product of an incoming vector $\mathbf{v}$ (matrix in the batch case) with the softmax Jacobian: $\mathbf{z} = \mathbf{v} * J_{\mathbf{x}}\boldsymbol{\sigma}(\mathbf{x})$. Show that this product can be efficiently implemented as $z_i = \sigma_i(\mathbf{x})(v_i - \mathbf{v} * \boldsymbol{\sigma}(\mathbf{x})^\intercal)$

Solution:

4.1c)

For stability reasons, we want to augment the softmax implementation by subtracting $\max_{i=1,\dots,C} x_i$ from the input such that all values passed to the exponential functions are negative. Please prove the statement that $\sigma_i(\mathbf{x}) = \sigma_i(\mathbf{x} + c)$ for $c \in \mathbb{R}$.

Solution:

4.1d)

Derive an expression of the cross-entropy's Jacobian $J_{\mathbf{z}}\mathcal{L}(\mathbf{z},\mathbf{t}) \in \mathbb{R}^{1\times C}$ where $\mathcal{L}(\mathbf{z},\mathbf{t}) = -\sum_{i=1}^{C} t_i \ln(z_i)$. The target $\mathbf{t} \in [0,1]^{1\times C}$ of $\mathbf{z}$ follows $\sum_{i=1}^{C} t_i = 1$.

Solution:

4.1e)

Show that the derivative of tanh($\mathbf{x}$) can be computed as $\frac{\partial}{\partial \mathbf{x}}\tanh(\mathbf{x}) = 1 - \tanh^2(\mathbf{x})$.

Solution:

Task 4.2: Neural Network Implementation

In this task, you will implement a simple neural network from scratch for handwritten digit recognition (classification). We will start with only a Linear Layer and SoftMax activation function. Afterwards, we will implement a deeper neural network by stacking two linear layers using the tanh activation function.
We already provide you with a simple framework to build, train and evaluate your first neural network. We have already taken care of all the functionalities needed to train and evaluate your architecture. It is now your task to build up a neural network from scratch and making sure that parameter initialization, forward propagation, backward propagation and parameter updates work.
Please get yourself familiar with the provided framework before starting the implementation. We have created a readme for setting up the code base and getting started. Note that you are only allowed to use the already imported libraries!

Please also upload your written code solutions with clarifying comments to Moodle.

4.2a)

Within nn_modules.py, implement the LinearModule by following the abstract methods of the NNModule. Use the classical Xavier Glorot parameter initialization.

Solution:

4.2b)

Within nn_modules.py, implement the forward and backward pass of the SoftMaxModule.

Solution:

4.2c)

Within nn_modules.py, implement the CrossEntropyLoss by following the abstract methods of the LossModule.

Solution:

4.2d)

Within nn_model.py, you can find the class NNModel which should provide all the needed functionalities to train and evaluate a generic stack of NNModules. Please implement the methods for initialization, forward and backward propagation as well as parameter updates.
You are now ready to train and evaluate your first neural network. Run the python script main.py to check your implementation. The predefined architecture consists of a Linear(28*28, 10) layer with SoftMax activation.

Solution:

4.2e)

We now want to build a deeper neural network. Within main.py uncomment the deeper neural network module specifications (Linear(28*28,200), Tanh, Linear(200, 10), SoftMax). Please implement the forward and backward pass of the TanhModule inside nn_model.py. Train and evaluate your deeper neural network by again running the main.py python script. Provide the loss curve of the training phase and report your final accuracy on the test dataset.

Solution:

Neural Networks with JAX

In this section, we ask you to familiarize yourself with JAX and Flax. Feel free to use any resources you find on the internet. Here is the ones we recommend:

- for getting into JAX: `https://jax.readthedocs.io/en/latest/notebooks/quickstart.html`
- for getting into Flax: `https://flax.readthedocs.io/en/latest/guides/flax_basics.html`

We ask you to use a CPU and not a GPU.

Task 4.3: Start by answering the coding questions on the Jupyter notebook "neural_network_jax.ipynb". Each question is also worth 1 point.

We recommend installing the dependencies with the following command line: "pip install -r requirement.txt" in a Python virtual environment with Python 3.8.
Please hand back the filled jupyter notebook.

Task 4.4: Building intuition on the neural networks

This task uses the jupyter notebook that has been filled up in the previous questions.

4.4a)

Set LEARNING_RATE to 0.001, N_SAMPLES to 500, USE_CONVOLUTIONS to False and RELU_AS_ACTIVATION to True. Report the plot of the training and testing metrics obtained from the jupyter notebook. Report the plot of the first miss-classified sample of the test dataset. Report the training time as well. Comment on the results.

Solution:

4.4b)

Set LEARNING_RATE to 0.001, N_SAMPLES to 500, USE_CONVOLUTIONS to False and RELU_AS_ACTIVATION to True. Uncomment the 3 lines: "# @partial(jax.jit, static_argnames="self")" above the functions "update_model", "loss" and "accuracy". Compare the new training time with the one of the previous question. Comment on the results.
We ask you to leave the 3 lines uncommented until the end of the homework.

Note: if the code is throwing an error, it means that you have implemented an operation not compatible with the "jax.jit" function. Please change your implementation to make the jitting possible.

Solution:

4.4c)

Set LEARNING_RATE to 0.001, N_SAMPLES to 500, USE_CONVOLUTIONS to True and RELU_AS_ACTIVATION to True. Report the plot of the training and testing metrics obtained from the jupyter notebook. Report the plot of the first miss-classified sample of the test dataset. Report the training time as well and compare it to the one of the previous question. Comment on the results.

Solution:

4.4d)

Set LEARNING_RATE to 0.001, USE_CONVOLUTIONS to True and RELU_AS_ACTIVATION to True. Report the plots of the training and testing metrics obtained from the jupyter notebook with N_SAMPLES set to 500, 1000 and then 2000. For each value of N_SAMPLES, report the plot of the first miss-classified sample of the test dataset. Comment on the results.

Solution:

4.4e)

Set N_SAMPLES to 500, USE_CONVOLUTIONS to True and RELU_AS_ACTIVATION to True. Report the plots of the training and testing metrics obtained from the jupyter notebook with LEARNING_RATE set to 0.00001, 0.001 and then 1. For each value of LEARNING_RATE, report the plot of the first miss-classified sample of the test dataset. Comment on the results.

Solution:

4.4f)

Set LEARNING_RATE to 0.001, N_SAMPLES to 500, USE_CONVOLUTIONS to True and RELU_AS_ACTIVATION to True. Comment the 2 lines where you have implemented average pooling in the "__call__" function of the CNN class. Report the plot of the training and testing metrics obtained from the jupyter notebook. Report the plot of the first miss-classified sample of the test dataset. Report the training time as well and compare it to the one obtained when average pooling was used. Comment on the results.
For the following question, uncomment the 2 lines where you have implemented average pooling.

Solution:

4.4g)

Set LEARNING_RATE to 0.001, N_SAMPLES to 500 and USE_CONVOLUTIONS to True. Run all cells with RELU_AS_ACTIVATION set to False and set to True. Set RUN_BOTH_ACTIVATIONS to True in the last cell of the jupyter notebook. Report the plot of the norm of the gradients. Comment on the results.

Solution: